Spring 5-1-1993

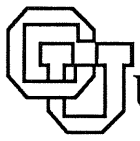
Increasing Shared Understanding of a Design Task Between Designers and Design Environments: The Role of a Specification Component ; CU-CS-651-93

Kumiyo Nakakoji
University of Colorado Boulder

Increasing Shared Understanding of a Design Task Between Designers and Design Environments: The Role of a Specification Component

Kumiyo Nakakoji

CU-CS-651-93 May 1993



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Increasing Shared Understanding of a Design Task Between Designers and Design Environments: The Role of a Specification Component

by

Kumiyo Nakakoji

B.S., Osaka University, Japan, 1986

M.S., University of Colorado, Boulder, 1990

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

1993

Nakakoji, Kumiyo (Ph.D., Computer Science)

Increasing Shared Understanding of a Design Task Between

Designers and Design Environments: The Role of a Specification Component

Thesis directed by Professor Gerhard Fischer

Abstract

Design is ill-defined and open-ended. Traditional waterfall process models do not support the way in which specifying a design problem and constructing a solution are intertwined. Knowledge required for a design task can neither be completely identified nor formalized before generating a solution that fits the task's changing needs. Integrated, domain-oriented, knowledge-based design environments are computer systems that provide design tools and information repositories that help designers understand and frame their design task. The KIDSPECIFICATION component supports designers in informally specifying their design goals, objectives, criteria, and constraints in the KID (Knowing-In-Design) design environment for kitchen floor plans. Explicit knowledge representations embodied by the specification component, in conjunction with a construction component that supports designers in constructing floor plans, provide *"shared understanding"* of the task at hand between human designers and the design environment. The increased shared understanding by the specification component augments the quality of knowledge delivery mechanisms of KID that deliver the design knowledge relevant to the designers' task at hand. Protocol analyses of designers interacting with KID demonstrated that the explicit representation of designers' goals and intentions helped designers in better understanding a design problem by allowing them to (1) *frame* concrete objects to reflect on as a partial specification and construction, and (2) *see* the partially framed design with task-relevant design knowledge delivered by KID. This seeing-framing-seeing cycle of design processes driven by designers in collaboration with the design environment has been frequently observed. Designers' reflection was enhanced by the delivery of sometimes unexpected information. The delivery encouraged designers to articulate design knowledge that had been implicit. Assessment of these mechanisms illustrates the beneficial role of the specification component in conjunction with the knowledge delivery mechanisms in enabling the KID design environment to effectively support human designers.

Acknowledgments

There are a number of people who deserve my thanks for their support while I completed this work.

First of all, I would like to thank my employer, Software Research Associates Inc., Tokyo, Japan, who gave me a chance to study abroad and pursue this doctoral degree, and most of all, allowed me to enjoy being in an "ivory tower" for five years with their generous financial support. In particular, Koichi Kishida, a vice-president and a technical director of the company, has been constantly encouraging, understanding, and constructively critical in my research. Masatoshi Matsuo and Kaoru Hayashi have been excellent managers who have taken care to provide the best conditions to complete my research. Hiroshi Sakoh, my senior colleague, has been my model as a researcher, with his enthusiasm toward science. Atsushi Aoki and Yoshiyuki Nishinaka, other colleagues at the SRA Boulder office, have been supportive. I thank Koji Torii, my former advisor at Osaka University, Japan, who talked me into going back to school and getting a degree. I am extremely grateful for his continuous encouragement. I thank Kenichi Matsumoto, who has been very understanding and supportive and who guided me in finishing a degree.

I thank the dissertation committee members, Wayne Citrin, Mike Eisenberg, Mark Gross, Dennis Heimbigner, James Martin, and Raymond McCall, who gave me priceless comments from various perspectives and helped me to improve my dissertation. I would like to give very special gratitude to Gerhard Fischer, my advisor. He not only generously let me share his deep knowledge and insights in the research field, but also patiently listened to my sometimes rough ideas and guided me in a research direction. It was he who taught me the joy of doing research as well as how to enjoy life, and continuously encouraged me with kind consideration and humor. Without him as my advisor, I could not have enjoyed completing my dissertation as much as I did.

I thank Barbara Gibbons of Kitchen Connection at Boulder, Colorado, for her valuable time and commenting on my work. She generously offered us her domain knowledge and expertise as a very successful professional kitchen designer. Her positive feedback as well as criticisms were very helpful to improve the design of KID.

On the Boulder campus of the University of Colorado, students in our Human-Computer Communication research group were helpful, patient, and very supportive of my research. I would like to give special thanks to Jonathan Ostwald, the first person in the research group to talk to me when I first came to Boulder in 1988, who has been always there to help me. He gave me tremendous support in organizing research ideas, correcting my English compositions, as well as learning the American culture. Without his support, I could not have completed my dissertation in this manner. I thank Anders Morch, Andreas Lemke, and Andreas Girgensohn, who provided the foundations for this research in terms of design environments, and Scott Henninger, who helped me to develop the idea of knowledge delivery. I was extremely delighted to have a wonderful office-mate Brent

Reeves, who gave me tremendous emotional support. It was a great pleasure that I could work with him in completing our dissertations together while pushing science forward. I thank David Redmiles, who generously offered his precious time in helping me and cheering me when I got stuck. I thank Tamara Sumner and Gerry Stahl, who read a draft of my dissertation very carefully every week in the last few months and encouraged me to keep going with very constructive criticisms over *water-of-life*. Alexander Repenning and Nadia Repenning have always been there to give me intellectual, emotional, and nutritious support. I thank people who gave me comments on a demo of my system, on the draft, and during the user studies, including Ernesto Arias, Alexander Wolf, Lloyd Williams, and John Rieman. I would also like to thank Francesca Iovine and Dorothy Foerst, who helped me in preparing my dissertation.

Finally, I would like to thank my family, friends, and relatives who all endlessly encouraged and supported me from across the Pacific Ocean. Especially, I thank Sueno Kawanishi, my grandmother, who always wished me good luck, and Hiroko Yamashina Arai, a friend who has seen me though good times and bad. Lastly, but most of all, I would like to thank my mother, Sachiyo Nakakoji, and my late father, Noboru Nakakoji, who taught me the joy of study twenty-nine years ago. I would like to dedicate my whole work to Sachiyo.

Contents

Tables

Figures

Figure

Chapter 1

Introduction

Design has two characteristics: ill-definedness and open-endedness. *Ill-definedness* means that there is no optimal solution to the design task, but only a solution that "*satisfices*" [Simon 81]. This characteristic implies that one cannot specify a design problem completely before starting to solve it; designers have to gradually refine both the problem specification and the design solution at the same time. *Open-endedness* means that knowledge necessary for a design can never be completely articulated a priori. It is neither possible to identify all the relevant knowledge for the design nor to formalize it for generating a design that fits to varieties of changing needs of design tasks.

The research presented in this dissertation has taken a human-computer cooperative problem-solving approach to support design activities. The approach augments the skills of human designers with *integrated, domain-oriented, knowledge-based design environments* instead of generating solutions for designers as typified by the design automation approach. Design environments are computer systems that provide design tools and information repositories that designers use for understanding, reflecting on, and forming their designs. In this approach, a computer system becomes a collaborative assistant and an intelligent agent for designers.

The research explores the role of a specification component in the KID (Knowing-In-Design) design environment for kitchen floor plans. The specification component supports designers in framing their design problems — that is, specifying design goals, objectives, criteria, or constraints. The specification component, in conjunction with a construction component that supports designers in constructing floor plans, embodies knowledge representations of the task at hand to be shared between human designers and the design environment.

The specification component enables an explicit representation of the designer's goals and intentions with respect to the current design. This explicit representation serves two purposes:

1. **Benefits for designers themselves:** using the specification component, designers are encouraged to articulate their design problems. Thus, coevolution of problem specification and solution construction is supported because designers can concurrently reflect on explicitly represented partial constructions and specifications.
2. **Benefits from the design environment:** information given through the specification component increases the shared understanding about the designers' intentions for the current task with the design environment. Thus the design environment can deliver task-relevant information for the designers' perusal.

An advantage of this approach is that increasing quality of information delivery can be achieved at no cost. The design environment delivers task-relevant information for designers without asking

Table 1-1: The Main Theme of the Dissertation

	Ill-definedness	Open-endedness
Problems	Inadequacy of models that separate problem analyses from solution syntheses	Information overload
Goals	Support coevolution of problem specification and solution construction	Tailoring information space
Theoretical Approach	*Seeing-Framing-Seeing* Cycle	Knowledge delivery
Prerequisites	Articulation of problem specification and solution construction	Shared knowledge representation about the task at hand
Mechanisms: RULE-DELIVERER CASE-DELIVERER	- Reasoning by argumentation Case-based reasoning	- Delivery of argumentation (through *specific critics*) Delivery of catalog examples

---

them for any additional efforts except to specify a problem using the specification component. Because specification of a problem is an essential activity in design, designers do not perceive any additional cost but only benefit in using the specification component.

In this research, I prototyped and studied several *knowledge delivery* mechanisms for the KID design environment. A knowledge delivery mechanism is one through which the right knowledge, in the context of a problem or a service, is delivered at the right moment for a human designer to consider [CSTB 88]. Assessment of these mechanisms illustrates the specification component's beneficial role in helping the KID design environment to more effectively support human designers.

Table 1-1 summarizes the theme of the research. The challenge in building effective design environments is coping with the two design characteristics: ill-definedness and open-endedness. Problems, goals, theoretical approaches, and prerequisites are discussed for the two characteristics. The research demonstrates that having a specification component in a design environment answers both to ill-definedness and to open-endedness. Two knowledge-delivery mechanisms, RULE-DELIVERER and CASE-DELIVERER, are described in the context of KID. The delivered knowledge, which includes design principles in a form of argumentation by RULE-DELIVERER and case-based information by CASE-DELIVERER, supports designers to exploit the system's stored design knowledge for reflection on their partial design.

1.1. Design

Design is Ill-Defined. In many situations, people are initially unable to articulate complete requirements for problems [Fischer, Reeves 92]. Current goals lead to a partial solution, while the gradually changing overall solution suggests new goals [Simon 81]. Professional practitioners have at least as much to do with defining the problem as with solving the problem [Rittel 84]. Problem framing and solving cannot be separated.

This implies that any type of specification-driven approach cannot be successfully applied to design. A design model that separates problem analyses from solution syntheses [Cross 84], such as the waterfall, model is inadequate because it requires problem specification to be completed before starting to form a solution. Any kind of formal system that automates design processes will be unsuccessful because it is based on the assumption that a problem specification never changes once solution development has started. Designers have to alternate between specifying a problem and constructing a solution.

Understanding what the problem is plays a major part in design activities. Starting with a vague incomplete problem requirement, designers sketch out a partial solution. By seeing the partial solution, designers identify portions of the problem that have not yet been understood, gain an understanding of the problem, and then refine the solution [Snodgrass, Coyne 90]. By iterating this reflection, understanding of the problem gradually emerges. This process characterizes design as a *reflective conversation* [Schoen 83] with the materials of design construction.

To support the reflective conversation with a design environment, I propose a model of the *"seeing-framing-seeing"* cycle of design processes, which is an extension of Schoen's *seeing-drawing-seeing* cycle [Schoen 83]. With the seeing-framing-seeing cycle, *framing* includes *both manipulation of explicitly represented problem specification and solution construction.* In the seeing-drawing-seeing cycle, Schoen defined "seeing" as reflecting on "drawing," a partially drawn solution in terms of the *tacit* problem specification. However, designers must not only see the partial solution in terms of the partially stated problem specification, but also *"see"* the partial problem specification itself. Seeing is evaluating and appreciating a design situation, which embodies both the partially framed problem and solution.

Support of this *seeing-framing-seeing* cycle in design environments requires tools that allow designers to articulate their present context; that is, both problem specification and solution construction. A CAD (Computer-Aided Design) system that only supports construction of solution forms cannot help designers to reflect on their partial design effectively because there is no explicit representation in the system of the problem they are attempting to solve. A design rationale recording system such as gIBIS [Conklin, Begeman 88] supports designers only in defining a problem; therefore, it cannot help designers effectively either because the information provided in the system is not directly related to a partially constructed solution.

A partially represented problem specification and a partially constructed solution in a design environment represent the designers' problem context. The designers' "task at hand" is defined as this partially identified context throughout the dissertation.

Design is Open-Ended. Design environments can make a large amount of complex design knowledge available to designers. Ironically, this increases the complexity of design problems. Often designers are unable to articulate what information they need. They may be unaware that they need information or that useful information exists in the system. Designers make limited use

of stored information because of the large and growing discrepancy between the amount of potentially relevant information and the amount any one designer can handle [Draper 84].

To cope with this problem of *information overload*, design environments must tailor the designer's information spaces according to the designer's task at hand. *Knowledge delivery* is a mechanism that detects the potential for a design information need and presents stored knowledge for the designer's perusal.

Knowledge delivery mechanisms require design environments to maintain information about the designer's task at hand. Sharing the context in which the task at hand evolves enables a design environment to detect a designer's information needs, and make the information space relevant to these needs.

1.2. Human-Computer Cooperative Problem-Solving Approach

The above discussions have led us to take a human-computer cooperative problem-solving approach to support designers. Automated design approaches are inadequate because designers cannot articulate a problem before starting to solve it. Expert system approaches are inadequate because of the open-ended nature of design. We cannot construct a knowledge base that will not break down [Winograd, Flores 86].

Current research has focused on two types of human-computer cooperative problem-solving approaches. In the first, computer systems create solutions and designers critique them and guide the solution process. In the second, human designers produce partial solutions and systems provide relevant information and critique them.

This research focuses on the latter — to build a design environment that augments skills of designers. People enjoy *doing*, and are good at creative activities. Although there is no reason to prevent computers from producing a solution when possible, designers must have control over accepting or rejecting it. Finding the proper balance in distributing roles and control between designers and computer systems is the challenge in realizing an effective human-computer cooperative problem-solving system.

In human-human cooperative problem solving, participants compliment each other with their own skills and knowledge by trying to increase shared knowledge about their problem and context through communication. How then can we apply this paradigm to human-computer cooperative problem solving?

My claim is that a specification component that allows designers to specify an abstract design requirement helps to establish shared understanding between a design environment and designers. This research explores the role of the specification component in a design environment. *"What is problem specification?"* and *"How can it be used?"* are the questions addressed by this dissertation.

1.3. Framework: Domain-Oriented Design Environments

To provide construction and specification tools and information repositories for design knowledge, design environments must be *domain-oriented.* By closing the transformation gap between a design substrate and an application domain, design environments should make designers feel that they are interacting with a design domain rather than with low-level primitives. The computer should be invisible, by supporting communication between humans and their problem-domain rather than human-computer communication [Fischer, Lemke 88].

A **multifaceted architecture** has been developed by building several prototype domain-oriented design environments for various design domains. KID, an integrated knowledge-based domain-oriented design environment based on the multifaceted architecture for designing a kitchen floor plan, has been built and studied. JANUS [Fischer, McCall, Morch 89], a predecessor of KID, featured:

- the JANUS-CONSTRUCTION component, which provides a palette of design units that designers can pick up and combine in a work area using direct manipulation;
- the JANUS-ARGUMENTATION component, in which design knowledge is structured in a form of the IBIS structure (Issue-based Information Systems [Conklin, Begeman 88]); and
- a *Critics* mechanism, which identifies violations of general design principles in configurations of design units and notifies designers with critic messages.

KID extends the JANUS environment with

- the KIDSPECIFICATION component, which supports designers in specifying their design requirement, such as objectives, criteria, or constraints; and
- the CATALOGEXPLORER component, which supports designers in searching the catalog of prestored designs in terms of their task at hand.

KIDSPECIFICATION has enabled KID to cope with the ill-defined and open-ended characteristics of design. KIDSPECIFICATION allows designers to make their problem specification explicit by helping them to articulate what would otherwise remain tacit or unknown. A partial specification and construction represents an evolving artifact in KID. The evolving artifact provides representation of shared understanding about the current task at hand. Thus, by taking advantage of the presentation of an evolving artifact, KID provides two knowledge delivery mechanisms:

- RULE-DELIVERER identifies *specific critics* that notify designers of potential breakdowns in the partial construction in terms of the partial specification, and provides links to stored design principles in the argumentation base that relate to the current specification; and
- CASE-DELIVERER produces catalog examples ordered in accordance with the current specification.

Delivered knowledge supports designers in developing their design in KID in the seeing-framing-seeing cycle of processes. A specific critic that has been fired provides designers with linkage into the argumentation base in which the related argument is stored. KID also delivers catalog examples

that provide designers with case-based information, that is, potential solutions that other designers have built while solving similar design problems as well as warnings of potential failures. With these two types of design knowledge, designers can *''see''* their partial specification and construction using argumentation-based reasoning and case-based reasoning, and can identify a direction to proceed to *''frame''* their partial design.

1.4. System-Building Efforts

KID's major extensions over JANUS include the KIDSPECIFICATION component, the CATALOGEXPLORER system component, and its RULE-DELIVERER and CASE-DELIVERER mechanisms. All the components and mechanisms in KID are firmly integrated, and they communicate via *Specification-linking rules.*

Implementation of KIDSPECIFICATION. Professional kitchen designers often use questionnaires to elicit clients' requirements for their design tasks. The user interface design of KIDSPECIFICATION is similar to the questionnaire forms and is implemented as a hypertext interface to the argumentation base. The argumentation base of KID stores design rationale based on the IBIS [Conklin, Begeman 88] structure, which consists of issues, answers, and arguments. The argumentation-base of the domain provides a semi-structured design space containing considered questions, possible design alternatives, and reasonings as arguments. Selecting such alternatives can be viewed as articulating a problem. KIDSPECIFICATION provides prestored questions (issues) and associated optional answers, and designers select the concerns associated with their current design task. If no prestored alternatives express their position, designers can add or modify information in the underlying argumentation base. Designers can assign weights to the selected answers to represent the relative importance of the specified items.

Specification-linking Rules. To deliver knowledge relevant to a partial specification, the system must know how to determine relevance. That is, the information represented in KIDSPECIFICATION must be linked to other types of information. *Specification-linking rules* represent the interdependency among the design decisions represented in a partial specification and a partial construction. In KID, specification-linking rules are used by KIDSPECIFICATION and the two embedded knowledge delivery mechanisms:

- KIDSPECIFICATION makes suggestions on how to develop the current specifications (which answer to select), and allows designers to play *what-if* games by providing suggestions that respond to a particular design specification.

- RULE-DELIVERER determines which critic rules should be activated as ''specific critics'' based on the current specification. Some specification-linking rules are related to features in the construction. The rules, then, can be used to identify applicable critic rules when certain design decisions are made in the specification.

- CASE-DELIVERER computes conformity of each catalog example to the current partial specification. CASE-DELIVERER applies a set of identified specific critics to each catalog example, computes an appropriateness value for the example as the weighted sum of the critic evaluations,

orders the examples according to the values, and presents the ordered catalog examples to designers.

CATALOGEXPLORER. CATALOGEXPLORER is the system component that supports designers in searching the catalog space in terms of their current task. A catalog example in KID contains two representations: (1) a specification, which is a set of selected answers in KIDSPECIFICATION; and (2) a construction, which is a floor plan constructed in CONSTRUCTION. CATALOGEXPLORER allows designers to locate useful catalog examples by searching for examples that partially or entirely match the current specification or construction.

1.5. Observed Benefits

KID has been studied by observing several subjects, including both domain-experts and novices in using the system. When design knowledge (either argumentation through a specific critic, or ordered catalog examples) is delivered, subjects have been observed to respond in the following three ways: (1) to apply the delivered knowledge to reframe their partial design, (2) to explore the related information space to the delivered knowledge, or (3) to articulate new design knowledge by arguing against the delivered knowledge.

The KIDSPECIFICATION component enables the KID design environment to have more shared understanding of a designer's task at hand. Consequently, KID supports reflection in action during a design process by delivering the design knowledge relevant to the task at hand, and this reflection drives the coevolution of problem specification and solution construction.

1.6. Organization of the Dissertation

In Chapter 2, I discuss theories in design. I first describe what I mean by *specification* and *construction* of design by comparing the vocabularies used in different design disciplines. Then I describe the ill-defined and the open-ended nature of design in more detail by discussing how problem specification and solution construction are intertwined, and how design knowledge is tacit. Then, as an approach to challenge this nature of design, I propose a model of the *seeing-framing-seeing* cycle.

In Chapter 3, I discuss how computers can support designers based on the model of the *seeing-framing-seeing* cycle. Required supports for *framing* are discussed and the knowledge required for *seeing* is identified. In order to use the stored knowledge effectively, a design environment has to deal with the problem of information overload. I introduce knowledge delivery as an approach to this problem.

In Chapter 4, the multifaceted architecture for domain-oriented design environments is described. After a brief synopsis of the historical background, the components of the architecture are described in terms of the KID design environment. Two scenarios in Chapter 5 illustrate how KID surpasses JANUS with the extensions developed.

Chapter 6 discusses conceptual description, meanings, and the role of a specification component of a design environment. Then, the design of the KIDSPECIFICATION component is presented. *Specification-linking rules* are used to integrate KIDSPECIFICATION with the other the components and the delivery mechanisms in KID. I describe how the rules are derived automatically from stored design rationale in the argumentation-base, which provides the underlying information structure for KIDSPECIFICATION. In Chapter 7, a brief system description of KID is provided and two knowledge delivery mechanisms in KID, RULE-DELIVERER and CASE-DELIVERER, are explained.

In Chapter 8, results of user observations are reported. The results are illustrated with transcripts of videotaped sessions, which have been analyzed using a constructive-interaction method. The results show that designers are empowered in performing the seeing-framing-seeing cycle of design processes by KIDSPECIFICATION and the two knowledge delivery mechanisms.

Although comparison to related work has been discussed in other places relevant to each topic, Chapter 9 presents discussions of related work particularly focusing on five topics: knowledge-based design environments, computational critics, case-based reasoning, adaptive agents, and reusable design rationale.

Chapter 10 discusses the applicability of the approach to other domains, especially to software design. Chapter 11 concludes the dissertation.

Chapter 2

Theory of Design

In this chapter, I discuss design as a problem framing and problem solving activity. First I describe how design problems are ill-defined; a problem specification and a solution construction are intertwined. Second, I discuss how the space of design knowledge is open-ended; some design knowledge is tacit and continually being discovered by designers.

Design is to understand the situation that designers are in. The design situation is the context in which the designers are engaged with a partially framed problem and background knowledge, having a partially framed solution at hand. In order to identify the parts, which represent the design solution, we need to know the context, which includes specifying the problem. And in order to understand the context, we need to know the parts. This *Hermeneutic Circle* [Stahl 93; Snodgrass, Coyne 90] describes the inapplicability of a water-fall type model for design and formal systems because they depend on the hypothesis that the context can be defined completely before knowing what the parts will be.

In order to deal with the ill-structured and open-ended nature of design, I propose a cycle of *seeing-framing-seeing* as a model of the design process in which framing includes not just constructing a solution but also specifying a problem. The model is an extension of the seeing-drawing-seeing cycle proposed by Schoen [1983], which states that designers are engaged in *reflection in action* throughout the design process.

2.1. Intertwining Problem Specification and Solution Construction

Design tasks, such as architectural design, mechanical engineering design, software design, writing, or music composition, require human creativity and judgment in indecisive circumstances. Design tasks are *ill-structured* [Simon 84] or *wicked* [Rittel, Webber 84] because: (1) there is no clear definition of the problem space or the solution space, (2) there are no stopping rules to judge whether the design is completed or not, (3) we cannot judge whether the solution is right or wrong, and (4) there are no definite criteria for testing a proposed solution.

Because different design disciplines use different terminologies for representing the same aspects of design processes, it is important to understand their relationships. Table 2-1 illustrates the different terminologies used by architectural design communities, mechanical design communities, and software engineering communities.

Every design discipline has identified design activities as consisting of (1) a process of identifying a problem, (2) a process of identifying a solution, and (3) enactment of the solution. Products of these processes are representations of the identified problem and solution. In this dissertation, I use

Table 2-1: Taxonomy in Different Design Disciplines

Activities and Products	Architectural	Mechanical	Software
Process to identify problem	Functional Design	Conceptual Design	Requirement Analyses
Product: *Specification*	Requirement Specification	Design Requirement	Requirement Specification
Process to identify solution	Design	Design	Design
Product: *Construction*	Design Description	Design Specification	Design Document
Enactment	Building	Manufacturing	Implementing

the term *''specification''* for the process of identifying a problem. *''A partial specification''* is a representation of a product produced during the specification process. The term *''construction''* is used for the process of identifying a solution, and *''a partial construction''* is a representation of a product produced during the construction process.

In design tasks, problem specification and solution construction are intertwined. The processes of specifying the problem and of constructing a solution are reflexive, because every transformation of the specification of the problem provides the direction in which a partial solution is to be transformed, and every transformation of the constructed solution determines the direction in which the partial specification is to be transformed. Designers may start with a few initial specifications such as goals, objectives, and constraints. Other goals, objectives, and constraints may emerge during design, and the original specification may be modified in transforming the problem to the next stage.

A new way of looking at design is the hermeneutic approach, in which design is an interpretative activity [Stahl 93; Snodgrass, Coyne 90]. Generally, designers come to the design situation with a partial understanding of what the designed artifact will be (a partial problem specification). As they begin to construct a partial solution to the design problem, they have a vague anticipation of the completed product as the whole. As they proceed with their interpretation and as their understanding of the problem increases through an interpretation of the parts of the partial solution, the anticipated whole is refined [Snodgrass, Coyne 90]. Thus, understanding a design task evolves through attempting partial solutions.

Snodgrass and Coyne [1990] characterized the design situation as tacit understanding. Designers are in the situation of performing a design task but cannot completely articulate what the problems

are. The design process is to gradually reveal tacit understanding, which is continually refined. Understanding is a cyclic process because the context (specification) and the part (construction) give meaning to each other. Thus, specifying a problem and constructing a solution depend on each other and have to be framed concurrently by designers.

2.2. Tacit Knowledge in Design

Alexander [1964] identified two types of culture in designing: *self-conscious* and *unself-conscious*. In self-conscious design, solution-construction is governed by explicitly represented rules and principles. Design activities, however, do not always proceed in this manner. There are some unself-conscious activities involved in design in which failure and correction occur side by side; there is no logical description between the recognition of a failure and the reaction to it. The rules are not made explicit but are revealed through the correction of mistakes, such as in oil-painting [Simon 81].

Some design knowledge is *tacit*. Designers know more than they can tell [Polanyi 66], they tend to give inaccurate descriptions of what they know, and they tend to be unaware of what they know [Schoen 92]. The report on the *''silence game''* by Schoen [1992] reveals the divergence of interpretation of design prototypes and the existence of tacit parts in design knowledge. In the silence game, a participant (interpreter) guessed underlying rules of building blocks that another participant (builder) constructed. It was surprising to discover how difficult it was for the interpreter to read the (intended) meaning of a prototype. It was clear that a given construction could be interpreted in terms of more than one rule. The builder sometimes discovered that he had embodied more rules in his construction than he anticipated. The study demonstrates Rittel's [1984] characterization of design artifacts as *''buildings do not speak for themselves.''*

Snodgrass and Coyne [1990] provided a long discussion comparing the design process with the interpretation of text. The meanings of words in a body of text depend on the context within which they occur; but the context is made up of the concepts embodied by the words to which the context in turn gives meaning. We cannot define a word precisely, because its meaning fluctuates depending on the situation in which it is used. Our ability to understand text depends on our ability to reduce the ambiguity of the individual words by placing them within the global context of the situation in which they are used.

In design, ''context'' is provided by the problem specification, and ''words'' are elements that form the constructed solution. As the language cannot be described in terms of words that can be combined in accordance with the rules of logic to form meaningful sentences, *design activity cannot be described, codified, and explained in terms of an algorithmic logic model* [Snodgrass, Coyne 90]. It is thus impossible to write any programs that can tell computers to do ''design.''

2.3. Inadequacy of Current Approaches

The above theoretical aspects of design lead to the identification of two characteristic features in design: ill-structured and open-ended nature. We cannot separate problem specifications from solution constructions. Human designers cannot completely specify the problem to solve before starting to solve it. This makes any specification-driven design process models infeasible, and it makes the AI-based approaches that presuppose a priori specification to automated design processes inadequate. Some design knowledge is tacit. Any expert systems that have been designed based on the assumption that we can completely codify the expertise of professional designers in a knowledge-base will not work. We cannot build a knowledge-based expert system that is complete.

In practical engineering fields, people have started taking seriously the ill-structured nature of design rather than blaming fluctuating specification on designers. As such, many are convinced that the first-generation design methodologies [Rittel 84] or traditional approaches of software design [Sheil 83] (such as the waterfall model), which require a problem specification to be complete prior to starting a solution construction, are not appropriate. In mechanical engineering, Kalay [1991] observed that the process of design consists of three major activities *performed iteratively*: (1) defining a set of objectives that the proposed actions should achieve, (2) specifying actions that, in the opinion of the designer, will achieve the objectives, and (3) predicting and evaluating the effects of the proposed actions to verify that they are consistent with each other and that they will achieve the objectives. In software engineering, Swartout and Balzer [1982] observed that implementation decisions are made before specification is complete, and these decisions can have a major effect on the further specification of the system. As a response to these observed problems, new design models that allow the intertwining of problem specification and solution construction have been introduced. Concurrent engineering [Young, Greef, O'Grady 91] integrates conceptual design and manufacturing, and the spiral model for software development [Boehm 88] allows a cycle of problem specification, design, and implementation.

These approaches, however, do not take into account the open-ended feature of design: the existence of tacit design knowledge. Because we cannot completely codify design knowledge, but only the parts that have been articulated, computer systems can have only a limited part of design knowledge in their knowledge bases. Computer systems cannot automate a design, and human designers, therefore, have to be actively involved throughout the design process.

In this dissertation, I suggest as a design process model a cycle of seeing-framing-seeing, which recognizes the ill-structured and open-ended nature of design. The model is based on Donald Schoen's [1983] *reflection-in-action*, which views design as a cycle of seeing-drawing-seeing.

2.4. Reflection in Action

When reading a sentence, we have initial anticipations and expectations of what the meaning of the whole sentence will be and interpret each word accordingly. Based on the interpretations of each word, we refine the anticipations of the meaning of the whole. The redefined whole functions in a dialectical fashion to refine and redefine the parts [Snodgrass, Coyne 90].

Design proceeds in the same manner. With Schoen's [1983] *seeing-drawing-seeing* model, designers *see*, *draw*, and *see* again engaged in a design world. Starting with a vague incomplete problem requirement, they sketch out a partial solution. By seeing the partial solution, designers identify portions of the problem that have not yet been understood, gain an understanding of the problem, and then refine the solution. By iterating this reflection, an understanding of the problem gradually emerges. This process characterizes design as a *reflective conversation* [Schoen 83] with the materials of design construction.

To support the reflective conversation with a design environment, I propose a model of the *''seeing-framing-seeing''* cycle of design processes, which is an extension of Schoen's *seeding-drawing-seeing* cycle [1983]. With the seeing-framing-seeing cycle, *framing* includes manipulation of *both* explicitly represented problem specification and solution construction.

With the seeing-*drawing*-seeing cycle, Schoen defines "seeing" as reflecting on "drawing," identifying patterns and giving meanings to parts in a partially drawn solution in terms of the problem specification. However, designers must not only see the partial solution in terms of the partially stated problem specification, but also ''see'' the partial specification itself. *Seeing* is defined as evaluating and appreciating a design situation that includes both the partially framed problem and the solution. Designers need to identify mismatches between the two, and this constitutes the design situation with a background. By reflecting on the design situation, designers refine both the problem specification *and* the solution construction. Understanding a problem and understanding a solution are complementary. Designers *frame* a partial problem and solution in explicit representations, and then the situation *talks back* to designers. Based on this feedback, designers refine the partial design. Reflective conversation is performed between designers and a design situation.

The seeing-framing-seeing cycle of processes are driven by designers and should not be governed by computer systems. Sharples [1993] described this cyclic process as a *rhythm* of enactment and reflection. The external activity (framing) is synchronized with internal thought (seeing), and the medium must afford the speed of the rhythms, which matches the pace of cognitive activity.

2.5. Summary

Design is ill-structured and some design knowledge is tacit. These two features lead to the following requirements for computer systems that support design. First, the systems have to support coevolution of problem specification and solution construction because designers cannot completely specify problems to solve before starting to construct a solution. Second, the systems have

to provide design knowledge but should not govern the designer's design process. Designers know more than they can tell. Articulated and stored knowledge in the systems should be used in such a way that this knowledge augments designers' ability instead of supplanting it.

I proposed a cycle of seeing-framing-seeing as a process model for design based on the notion of reflection-in-action. *How can computer systems then support this model?* Although we cannot completely account for design in terms of symbolic processing models, computers can be useful tools for manipulating descriptions of form, function, and behavior in such a way as to provide designers with convenient access to information [McLaughlin 92]. Computer systems can enhance the ability to recognize features of a new design situation that might not otherwise come to light.

In the next chapter, I describe the human-computer cooperative problem-solving approach and prerequisites for computer systems that support designers in carrying out the seeing-framing seeing cycle of design processes.

Chapter 3

How Computers Should Support Designers in the Seeing-Framing-Seeing Cycle

A notion of design environments has been developed based on the human-computer cooperative problem-solving approach [Fischer, Reeves 92]. Design environments augment skills of human designers instead of automatically producing designs for designers. Based on the theory of design discussed in the previous chapter, the following prerequisites have been identified for design environments necessary for supporting design based on the *seeing-framing-seeing* cycle:

- explicit representations of what to *frame* and *see*
- information about design knowledge that supports *seeing*

In this chapter, I discuss how such design environments can support the *seeing-framing-seeing* cycle. The cycle is driven by conversations between designers and a design situation that is constituted by a partially specified problem and a partially constructed solution. The cycle consists of iterative processes of (1) *framing* a task and (2) *seeing* a task. In this dissertation, the term *"task at hand"* is defined as this design situation.

In this chapter, I first emphasize the necessity for explicit representations of a partial construction and a partial specification. An *"explicit representation of a specification"* does not mean a formal statement but an informal statement about goals, objectives, and constraints of the task. A partially represented problem specification and a partially constructed solution in a design environment represent the designers' task at hand. Explicitly articulating the task at hand lessens the cognitive load of designers' memories and additionally could uncover some of the hidden tacit relationships of the task.

Next, I describe the functions of *seeing*. Seeing is appreciating and evaluating. Seeing is necessary when formal logic is not applicable. Designers have to *see* the task at hand by applying their design knowledge, such as heuristics and experiences, while taking into account the global context.

My view of seeing is reasoning that consists of discovery and validation [Simon 91]. I maintain that in order to support these processes, design environments need (1) to support designers in changing their conceptual focus, and (2) to provide design knowledge relevant to the task at hand. I describe what types of design knowledge are necessary in order to discover and validate, what types of representations are appropriate, and what issues enable designers to use the design knowledge. I claim that the "knowledge delivery" paradigm addresses the above two requirements. Knowledge delivery systems automatically provide designers with information relevant to the designers' task at hand. Because delivered design knowledge is in general associated with a certain conceptual view, the delivered knowledge also supports designers in changing their conceptual focus.

3.1. Explicit Representations of the Task At Hand

With the seeing-*framing*-seeing cycle, designers must be able to carry out reflective conversations with the task at hand, which is represented with a partial problem specification and a partial solution construction. Conventional CAD (Computer-Aided Design) systems provide explicit representations for a partial construction but not for a specification. When dealing with elements in CAD systems, designers have to listen to a back talk of the situation by *seeing* the explicitly represented partial construction in terms of the partial specification that only vaguely stays in the designers' mind as a mental plan or schema. Hypertext systems for recording design decision processes and design rationale, such as gIBIS [Conklin, Begeman 88], SIBYL [Lee 90], and DESIGNRATIONALE [MacLean, Young, Moran 89], support designers in framing the problem specification by keeping track of interdependency among design decisions made, but do not provide representations for the construction. In CAD systems as well as design rationale recording systems, the partially framed specification is detached from the partially constructed solution.

Externally articulating a partial specification is equally as important as externalizing these partial construction, and both of the representations should be integrated. When designers pause for reflection, they are directing attention toward an intimate relationship between the partial specification and construction. Putting ideas down on paper is not a matter of emptying out the mind but of actively reconstructing it, forming new associations, and expressing concepts in linguistic, pictorial, or any explicit representational forms while lessening the cognitive load required for remembering them [Sharples 93]. What something "means" lies in how it connects to other things we know [Minsky 91]. By providing links, relations, and connections between a represented partial specification and construction, designers gain an understanding of the partial design task.

When people explicitly represent an object, they discover alternative interpretations of the object and gain understanding of the object. Experimental results led Sharples [1993] to report that in writing, as the text emerges, the author can perceive it as both a writer and a reader, discovering new interpretations that may need to be captured quickly before they are forgotten. Another experiment, presented by Reisberg [1987], showed that when subjects are asked to memorize ambiguous figures and then draw pictures of their mental images, they are often able to see alternative interpretations they had not recognized before.

In order to successfully continue the coevolution of problem specification and solution construction, designers have to understand what the current task is. By externalizing the partial design task, designers gain new understandings that can then form and drive the next move in reframing the specification and construction.

The use of external representations helps free the designer of the burden of mental creation. Not only is the act of externalizing or the form of the external representations important, but also the close supportive relationship between the designers cognition and the external representations [Sharples 93].

Providing explicit representations of a partial problem specification may seem to contradict to the views discussed in the previous chapter; that is, that a design problem space is open-ended. However, the open-ended feature of design does not prohibit us from providing a *"partial"* representation of the problem space. First, even a partial representation will realize the benefits described above. Second, the open-ended feature only implies continuous growth of the space. An adequate support for *end-user modifiability* [Girgensohn 92] that allows designers to add and modify the representations for a problem specification will partially solve that problem. Representations of a partial problem specification and a tool to deal with them are further discussed in Chapter 6.

3.2. Design Knowledge Support for Seeing

Seeing as Reasoning. Designers sometimes do reasoning by seeing [Simon 91]. Simon presented as an example the observation that people agree that two diagonals in a square intersect not because they prove it using the axioms of Euclidean geometry, but simply because they *see* it. People have certain processors for operating on this information and drawing conclusions from it. We draw many inferences implicitly, by *seeing* them.

Reasoning by seeing requires designers to use heuristics, or design knowledge. Simon [1991] stated that a reasoning process has two parts, *discovery* and *validation*, that intermingle and alternate at all stages in our thinking. First, we discover new conclusions by using our past experiences and heuristics. Then, before we actually react to the discovered conclusions, we bring them to careful analyses to make sure the discovery is significant.

In summary, in order to support seeing, design environments need:

- to make designers aware of the existence of different points of view and to notify designers to change their conceptual focus for facilitating discovery, and
- to provide design knowledge, such as past experiences and heuristics, that maps a representation of the problem to the representation of the solution for facilitating validation.

Facilitating Discovery: Changing Focus. To facilitate discovery, systems need to support designers in changing a focus by providing them with other points of view. To discover a new feature in a partial design involves an understanding of the partial design. Miyake [1986] studied the cycle of *"non-understanding"* and *"understanding"* by observing two persons interacting while trying to understand the mechanism of a sewing machine in terms of a transition of conceptual points of view. Miyake observed that the subjects shifted their conceptual points of view more often when they were not understanding, and the shifts were often triggered by *critiques* from the other participant.

Miyake [1986] accounted for the role critics play in augmenting human understanding, viewing critics as the expression of validation checks from different points of view. Because individuals work on their own problems in their own ways, Miyake claims that they do not have easily acces-

sible checking mechanisms for the validity of their solutions from other points of view. When each participant works from a different starting schema, what is obvious and natural to one may not be so to the other. Miyake reported that self-criticizing accounted for only 12 percent of the incidents, implying that validation checking is indeed difficult to obtain within an individual system. Criticizing at the level at which both participants were working is also rare.

We need computer support for triggering this critiquing mechanism. Miyake observed that there are two types of critiquing: (1) *upward* criticism (experts criticize novices), and (2) *downward* criticism (novices criticize experts). An interesting observation was that downward criticism took place more often, and that these criticisms forced the other person, the one with more understanding, to try to understand the problem better. It seems, therefore, that criticism could occur when the two people were at different levels, and had different focuses. It is not simply that one person knows better than the other, or that both participants work on "the same level." Miyake's research showed that one observer does not necessarily have to be smarter than the other. This implies that in the human-computer cooperative paradigm, design environments need not necessarily be *smarter* (have more domain knowledge) than designers. A design environment can still criticize designers as a partner regardless of its amount of design knowledge.

Another of Miyake's observations was that the shifts of conceptual points of view had two directions: topic-related, and topic-divergent. Interestingly, topic-divergent shifts were most frequently initiated by novices and often worked constructively. This is related to the issue of *"focus."* If experts were more engaged with a local focus, Miyake hypothesized that novices had a more global focus, not yet being able to or not having to narrow their focus to match the experts. Topic-divergent shifts might have their origins in this global focus, which is not easily available to the experts. The novices can contribute by criticizing and giving topic-divergent shifts, which are not the primary roles of the experts.

Facilitating Validation: Evaluation. Seeing includes evaluation of the current design task. The meaning of the term *"evaluation"* in this dissertation has a much larger sense than the more rigid definition used by Kalay [1991]; *evaluation is the process whereby the possible characteristics or artifacts are simulated, hypothesized, or imagined, not necessarily actually archived.*

When designers evaluate a partial design, they use different criteria. Hinrichs [1990] described two types of unarticulated evaluation criteria used by designers when working with a partial specification and construction. *Criteria for completeness* is related to the question *"Am I done?" Criteria for consistency* includes consistency among parts of a construction, among different representations, and among partial specifications. These criteria are not well formed in any sense, but a part of design expertise.

Seeing relies on the designer's ability to: (1) *predict the environmental, psychological, social, economic, and other effects that will ensue from executing the specific actions,* and (2) *evaluate the desirability of these effects and derive operational conclusions from the evaluation* [Kalay 91].

Kalay stated that different levels of abstraction in design require application of different types of design knowledge. That is, heuristic rules will be used more frequently at the conceptual level, whereas precise procedural simulations might be necessary at the concrete structural level.

To evaluate the correspondence between partial specifications and partial constructions, designers must translate between the two different representational schemes, using design knowledge or the designers' expertise. Bonnardel [1991] observed 14 designers in evaluating aerospatial structures and found that experts were able to interpret (abstract) criteria in terms of (structural) constraints, whereas novices encountered difficulties in the interpretation. Criteria are represented in a partial specification and constraints are represented in a partial construction. Design knowledge is required to map the criteria into the constraints.

Types of Design Knowledge That Support Seeing. In order to support a mapping between representations for specifications and constructions, design environments need to provide two types of information: (1) heuristic rules that are accumulated via design rationale and are used to support argumentation-based reasoning, and (2) previously created cases that are used to support case-based reasoning.

For example, kitchen designers learn design principles through textbooks and training sessions. After initial training, these designers gain their expertise through practice. The designers identify new heuristic rules by solving specific design tasks. The heuristics help the designers to map structural features in construction to abstract features in specification. The following is a transcript made by observing a professional kitchen designer who has gained a new heuristic rule through a scenario.

> *Let's say that a kitchen is L-shaped, a dishwasher and a sink over here, and there is a stove on the other side, and two cooks in the kitchen. One is at the sink, cleaning up, and put dishwasher with the door down; then the other cook is at the range and he sort of forgets where he is and he backs up, and then he could actually trip over the dishwasher door and seriously injure himself.*

Once the designer identifies the heuristic rule that a dishwasher door should not interfere with the work space for a stove, she can identify the problematic structure in terms of the potential hazard for two cooks. This type of heuristics can be accumulated through recording a design rationale as a form of argumentation. The heuristic rules provide necessary conditions to the partial design task but are not necessarily sufficient [Simon 91]. Because human designers are in the loop of the seeing-framing-seeing cycle, such necessary-but-not-sufficient rules will be as useful as sufficient rules.

Kitchen designers also use catalogs. Catalogs are collections of previously constructed designs that represent what other designers have done. Catalog examples allow designers to make assumptions and predictions based on what worked in the past without having a complete understanding; they also provide a means of evaluating solutions when no algorithmic method is available for evaluation [Kolodner 90]. By seeing how other designers have solved the same type of subproblem, designers can gain the insight and ability to deal with their own design problems. This is a primary form of design experiences that makes a difference in performance between expert and novice designers.

3.3. Problems of Information Overload

In the second half of this chapter, I describe problems of information overload and suggest a knowledge delivery paradigm as a solution. Information overload means that the existence of huge amounts of design knowledge, or information, in the system does not necessarily imply a benefit for designers; it may result in increasing complexity of design problems [Fischer, Henninger, Nakakoji 92].

Designers comprehend representations of a partial design and state their desires in terms of their current task, context, past experiences, and views of the world. However, stored design knowledge is often based on the representation of information organized around someone else's view of the world (i.e., those who constructed the knowledge-base) [Fischer, Stevens 91]. Systems require designers to interact with representations with which they may not be familiar.

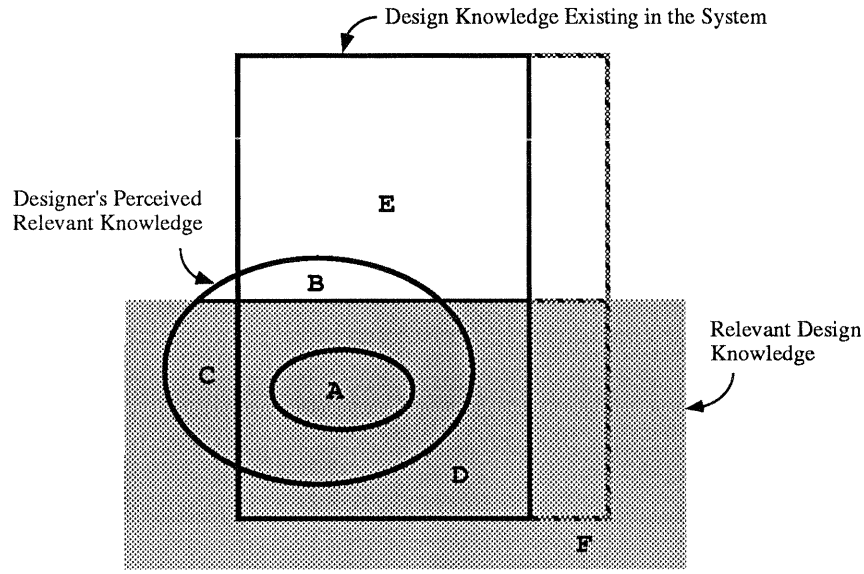
Information needs arise through a design task. Designers want to access information to solve a current design task; therefore, they should not be distracted too much, nor should they devote too much time in information retrieval. They should be able to retain the context of their current task.

Uses of design information, or represented design knowledge, in design environments have unique characteristics, some of which are different from those of other conventional text retrieval systems:

- Designers may not be able to articulate what they need.
- Designers may be unaware that they need information.
- Designers may not be motivated to search the information space because they are not aware of the existence of potentially useful information.

The information needed to understand a problem depends on the designers' idea for solving it [Fischer, Henninger, Nakakoji 92]. Designers cannot understand a problem without information about its context, but designers cannot meaningfully search for information without the orientation of a partial solution [Rittel, Webber 84]. In order to search effectively, designers must remind themselves of what they are looking for [Schank 88]. To do this, they must have a good understanding of the information space, which is gained by looking for information. Designers must know enough to ask a question; that is, to know what is not known [Miyake, Norman 79]. Looking for information is not a one-time task. After locating a piece of design knowledge, designers have to understand and modify it in order to apply it to their current task. In order to understand the knowledge, it should be *situated*, or associated in terms of the current design task [Greeno 89], which may require other information. This paradox makes conventional database retrieval techniques insufficient for locating useful information for a design task.

Existing database retrieval systems are built on the assumption that people searching for information know what information they need and how to ask for that information, but this assumption is often unwarranted [Fischer, Henninger, Redmiles 91a]. Pollack [1985] observed that in human-human cooperative problem-solving, people often do not know what information they need to obtain in order to achieve their goals, and consequently experts must identify inappropriate queries



Knowledge Contained in the Area	*Designers Think Relevant*	*Actually Relevant*	*Exists in the System*	*Notes*
A	**Yes**	**Yes**	**Yes**	
B	**Yes**	No	**Yes**	mis-conception
C	**Yes**	**Yes**	No	missing concepts
D	No	**Yes**	**Yes**	require delivery
E	No	No	**Yes**	
F	No	**Yes**	No	

Figure 3-1: Designers' Perceived and System's Actual Related Information Space
The small oval represents concepts designers know well and know how to access. Information in the large oval is what designers think is relevant to the task at hand, but may only vaguely know how to access.

---

and infer and respond to the goals behind them. Because many traditional information retrieval techniques are done in the context of text retrieval systems, their main emphasis is to provide access to many documents as efficiently as possible; little effort has been directed toward helping the user formulate a query [Thompson, Croft 89].

Designers are limited in making use of information because of the large and growing discrepancy between the amount of potentially relevant information and the amount any one designer can know and remember [Fischer, Henninger, Nakakoji 92]. Figure 3-1 illustrates this discrepancy.

In Figure 3-1, the rectangle represents the actual information space embodied in a system, and the gray area represents the information space that is relevant to the designers' task at hand at a particular point of time during a design process. The ovals represent the information spaces that designers think are relevant. The small oval (A) represents concepts designers know well and know how to access. Information in the large oval is what designers think is relevant to the task at hand, but only vaguely know how to access. Some of the information in the oval (the B area) may

be *misconceptions* of the designers [Silverman, Mezher 92] and might actually be irrelevant to the current task. If the designers try to access the information in area C, *missing concepts*, and realize that it does not exist in the system's knowledge base, they should be able to add that portion of knowledge to the system. Because designers do not recognize the existence of information in area D, they will never be motivated to look for it. It is noteworthy that if the system has more information (the larger rectangle with dotted lines) it is the area of D that increases, not that of the ovals.

As discussed above, traditional database retrieval systems support designers' use of information only in area A. New information retrieval mechanisms, including browsing, relevance-feedback, and retrieval by reformulation, have been proposed to allow designers to use the information in the large oval area [Fischer, Henninger, Nakakoji 92]. However, these mechanisms still do not permit designers to use the information in area D when they are not motivated to locate it because they do not know it exists.

Delivery mechanisms can provide the information in area D to designers without designers' explicit requests for information search. The delivery mechanisms deliver "*the right knowledge, in the context of a problem or a service, at the right moment for designers to consider*" [CSTB 88]. The mechanisms first detect the potential for a design information need and then present stored knowledge for the designers.

In the following subsections, I describe existing information access and delivery mechanisms as well as issues. *Information access* is the designer-initiated location of information when designers perceive their information needs. *Information delivery* is the system-initiated information delivery, inferred to be relevant to the designers' task at hand [Fischer, Henninger, Nakakoji 92]. This dissertation focuses on design and implementation of information delivery mechanisms in design environments using the shared knowledge about a design task provided by a partial specification and construction.

3.3.1. Discussions of Existing Access Mechanisms

Designers are often faced with situations in which they know that they need some information to solve their task at hand but cannot precisely articulate what they are looking for. Most of the existing information retrieval systems require users to map their information needs into a system model such as a formal notation using Boolean logic. Designers are not familiar with terms provided by the system, and therefore cannot select the necessary words to define a query precisely [Thompson, Croft 89]. In order to support designers in this situation, several approaches have been explored for helping users to express their information needs, including relevance feedback, browsing, retrieval by reformulation, and application of AI techniques.

Relevance Feedback. Typical relevance feedback methods ask users to rate the relevance of items retrieved by an initial query, and then use the ratings to reformulate the query, for example, by adjusting the internal representation of the query [Salton, Buckley 90]. There are several

problems with this type of approach. First, the technique puts a large burden on users because users may have to examine large numbers of retrieved items. Users must examine all the items to make sure that nothing relevant is missed [Thompson, Croft 89]. Second, effects on the query are invisible to users, and it is difficult to control the direction of the search because the invisible internal representation controls search direction.

Browsing. Browsing mechanisms provide an informal or heuristic search through a well-connected collection of nodes of information that provides designers with the information relevant to their task at hand, which they may not have been able to ask for by formulating a query. Designers determine the usefulness or relevance of the information currently being displayed in terms of the task at hand, and traverse its associated links if necessary with full control over the search while getting immediate feedback from the structure of the information space. Feedback from browsing differs from traditional relevance feedback in that the designers examine only one item at a time, evaluate its relevance, and select another item for view; it is the designer, rather than the system, who determines the items to be examined [Thompson, Croft 89].

One problem in applying this technique to stored design knowledge is that it is difficult to form a structure to ''well connect'' design knowledge. A space for design knowledge is open-ended and there is no single *right* structure that suits every design task. Another problem is that some design knowledge is not intuitively understandable and may require lengthy application to the current design in order to understand it. This loss of immediate feedback — the lengthy application of the knowledge to the current design — decreases the value of browsing. Third, when the information space is huge and complex, designers may easily get lost in a complex network of nodes while tracing dozens of links [Halasz 88]. Finally, it may be difficult for designers to identify initial nodes from which they can start traversing links. If they mistakenly start with an irrelevant node, there is no guarantee that they can get back to the relevant information space.

Retrieval by Reformulation. Retrieval by reformulation allows users to incrementally formulate a query by critiquing examples of items retrieved in a previous query. RABBIT [Williams 84] introduced the method by showing how a direct manipulation interface could be used in the process of query refinement, and the paradigm has been used in systems such as ARGON [Patel-Schneider, Brachman, Levesque 84].

An initial query retrieves a number of items, an example of which is displayed in the full form of the database representation. The user can then choose to require or prohibit specific parts of the example in reformulating the query. HELGON [Fischer, Nieper-Lemke 89] combines the retrieval-by-reformulation paradigm with a graphical hierarchical structure that allows designers to browse the database, and provides bookmarks for recording the history of information access.

Application of AI Techniques. People can set up situations and objects in the world so that the objects can be easily remembered in the future. The above discussed approaches provide access methods based either on a name space or a spatial metaphor. The approaches are inflexible because

the structures imposed may become obsolete, and effective navigation and location of old objects relies on the user maintaining an accurate model of the structure [Thimbleby et al. 90; Akscyn, McCracken, Yoder 88].

Several systems combine artificial intelligence (AI) techniques, such as semantic nets, frames, rules, and connectionism, with the conventional information retrieval techniques discussed above. The IIIR system (Intelligent Interface for Information Retrieval) [Thompson, Croft 89] uses a rule base in browsing. Rules are used to determine characteristics of a user's search, such as the expected number of relevant documents to be found, the expected number of formal searches that will be required to find the relevant documents, and the number of nodes to show in a neighborhood in browsing based on a user model and knowledge about the domain. Using the domain knowledge, IIIR can provide assistance to the user by giving advice on paths that should lead to relevant information.

LaSSIE [Devanbu et al. 91] provides a software library of reusable components that integrates architectural, conceptual, and code views of a large software system as a knowledge base. The index of reusable components of a large software system must reflect the programmer's view of the domain of application of the library. LaSSIE provides an intelligent indexing scheme that includes specific knowledge about the architecture within which these components are embedded. The system generates descriptions by reading and understanding the architecture documents and the comments in the source files, represents them in the KANDOR knowledge base, and uses them as indexes into the library. The system uses inference to answer the programmer's queries about reusable components.

CODEFINDER [Henninger 93] has further strengthened the query-by-reformulation paradigm by using spreading activation techniques, which automatically compute the strengths of association of keywords for information items. The approach overcomes a fixed index problem, in which it is impossible to predict how a given concept will be used for later retrieval. This was very problematic, especially in retrieving case-based information where defining indexes is a non-trivial matter [Wolverton, Hayes-Roth 91].

AQUANET [Marshall et al. 92] provides hypertext constructs with frame-based representation. Users can specify what its elements are and how they are interconnected, and they can modify and extend the knowledge structuring schemes as their understanding of the task changes, specify alternate graphic renderings of the same structure, compose knowledge structure, and negotiate about its contents. Being a "generic" tool, however, AQUANET puts much of the burden on users in performing knowledge-structuring tasks of defining semantics for nodes and links.

All of the above approaches for locating information require users to initiate the search; they have to be motivated to look for information. These approaches support users to access the design knowledge in the large oval area in Figure 3-1 and may unexpectedly bring them to access the information located in area D. However, as long as designers are unaware of their information needs, they will not start looking for information.

3.3.2. Discussions of Existing Delivery Techniques

When designers are neither aware of the existence of potentially useful information nor aware of their information needs, no attempt will be made to access the information. In these cases it would be helpful to have a system that monitors user behavior and delivers information relevant to the task in which the designers are currently engaged.

There have been several mechanisms that volunteer information to users without users' explicit requests. Most of these systems have little or no understanding of the users' task at hand. Thus, the retrieved information can be of little relevance to the current problem context.

Advice-Giving Systems. Owen [1986] described a system called DYK (''Did You Know'') in which the system presents unsolicited information about UNIX commands to the user. After presenting the information, the user is given the ability to browse around the topic area. DYK can effectively present information the user may have been unaware of. However, the information may not necessarily be from area D in Figure 3-1; it may also be from area E, unknown and irrelevant information. The random presentation of topics makes it even more difficult to understand why or how the information should be used because of the lack of the problem context.

ACTIVIST [Fischer, Lemke, Schwab 85] is an active help system for a text editor that infers user goals from observed actions by matching actions against plans in its knowledge base that accomplish the same goals. It dynamically maintains a user model containing information on how often a goal is accomplished, how often it is accomplished optimally, and how often the user is notified of the optimal method of achieving the goal. Because ACTIVIST does not have an explicit representation of the user goals, the system often *infers* rather naive suggestions.

Critics. Critics [Fischer et al. 91a] constitute another mechanism for delivering context-relevant information. They act as a set of rule-based agents that monitor the evolving construction of an artifact. When a problematic design situation is recognized, the critic presents some information to the user about what has been found.

The LISPCRITIC system [Fischer et al. 91a] allows programmers to request suggestions on how to improve their LISP code. The system proposes transformations that make the code more cognitively efficient (i.e., easier to read and maintain) or more machine efficient (i.e., faster or smaller). However, the lack of domain orientation limits the depth of critical analysis the critiquing system can provide. The system can answer questions such as whether the LISP code can be written more efficiently, but cannot answer whether the code can solve a specific problem.

Domain orientation in FRAMER [Lemke 89], a system for window-based user interface design on SYMBOLICS LISP machines, enables its critic mechanism to evaluate the completeness and syntactic correctness of the design as well as its consistency with interface style guidelines. Evaluations of FRAMER showed that many users did not understand why the advice was beneficial in solving their problem. Lemke has observed that when users do not understand why a suggestion is made, they

tend to follow the critic's advice without knowing whether or not it is actually appropriate to their situation. FRAMER II provided short explanations to address this problem, but it was difficult to provide concise and comprehensible explanations in the design domain.

JANUS [Fischer et al. 91a] makes a step toward addressing previous shortcomings by connecting the critic mechanism with argumentative discussions. JANUS contains two integrated subsystems: a domain-oriented kitchen construction kit and an issue-based hypermedia system containing design rationale. Critics respond to problems in the construction situation by displaying a message and providing access to appropriate issues. However, these critics often give spurious or irrelevant advice, resulting from the lack of an explicit representation of the user's task. The only task goal built into JANUS is one of building a good kitchen. With an explicit model of the designer's intentions for a particular design, critics can be selectively enabled and provide less intrusive and more relevant advice.

In the next section, I describe how design environments can support knowledge delivery mechanisms. I claim that the prerequisite for building effective delivery mechanisms of a design environment is to have a shared understanding between designers and the design environment about the designers' task at hand.

3.4. Challenges in Delivery

A problem for delivery mechanisms is that they have to work with incomplete knowledge about the designers' intentions. They may misinterpret the designers' goals because they lack some important knowledge and contextual information. Although the mechanisms for delivery can be designed and tailored for minimum disruption, a conflict will always arise between the need to inform users and the desire not to inundate them with irrelevant messages.

In human-human communication, both participants can adapt their own behavior according to the characteristics of the partners and by gradually gaining shared understanding. The communication process enriches and refines the knowledge of both partners about the task to solve and about each other. The shared understanding enables the partners to improve the communication process, to accelerate the discovery of either common or conflicting goals, to optimize the efficiency of the communication, and to increase the satisfaction of the partners [Oppermann 92].

In design environments, a partial problem specification and a partial solution construction can serve as a source of shared understanding about the design task. Design environments can monitor designers' activity as represented in the partial design, infer their information needs, search for relevant information, and present this information to designers.

Designers, on the other hand, have to have control over the delivery mechanisms. The delivery mechanism uses symbolic descriptions of how to infer the designers' information needs from a partial specification and construction, and how to search the information relevant to the identified task. As discussed in Chapter 2, such written descriptions can never be complete but are limited,

and there always exists a chance that the system may deliver irrelevant knowledge due to a situation that has never been anticipated.

Knowledge delivery provides *adaptivity* of the system, which automatically adjusts its behavior, information spaces, and functions to designers' task at hand, whereas knowledge access provides *adaptability* of the system, which designers can adjust. There is a spectrum in adaptivity according to the amount of control designers have. One end of the spectrum is *deterministic adaptations* of the application [Oppermann 92], which is a completely autonomous system. The other end of the spectrum is an *adaptable* system with which users have full control to adapt the system. In the middle is a *malleable* system [Fischer 92], which suggests adaptation to users who can then refine the adaptation for themselves.

To realize a more effective human-computer cooperative problem-solving paradigm, a design environment should be malleable; that is, it should provide both delivery and access mechanisms [Fischer, Henninger, Nakakoji 92]. Designers can make use of delivered information from the system and can take over the search by navigating through a space of proposed, possibly relevant, information spaces.

Challenges in realizing knowledge delivery mechanisms include how to deliver:

- the *right knowledge*,
- at the *right time*,
- in the *right style*.

The Right Knowledge. In order to deliver the right knowledge, design environments have to determine the relevance of information to the current task according to levels of the designers' skills. Research in adaptive systems [Oppermann 92; Kass, Stadnyk 92] uses two types of representation: a task model and a user model.

Having information from a partial specification and construction, knowledge delivery mechanisms of a design environment form a task model. Using this model, the delivery mechanisms can filter the irrelevant information from the knowledge base. However, determination of the relevant portions of knowledge is much more difficult. Different design situations may need to view a piece of knowledge differently. It is impossible to anticipate all possible design situations a priori [Suchman 87], which makes a static indexing scheme for design information of the knowledge bases inapplicable.

The partially identified current task can be used as queries submitted to information access mechanisms. Therefore, determining the right knowledge in terms of the partially identified design task can make use of research results from information access techniques such as spreading activation, a use of frames, or inference rules, as discussed in Section 3.3.1.

In addition to considering the relevance to the partially identified current task, knowledge delivery mechanisms must take into account degrees of designers' expertise. Evidence from information

science shows that differences among the information needs of individual users will affect the users' appreciation of the system [Frakes, Gandel 90]. For example, expert designers and novices require different levels of design knowledge to be delivered.

Finally, it is noteworthy that delivering *not-quite-right* knowledge to designers does not mean that delivery is useless. As will be discussed in Chapter 8, I observed during the user study that when designers found the delivered information to be of little or no relevance, they often were willing to modify the knowledge base by adding new design disciplines or concepts. Therefore, delivering knowledge to designers can be a *knowledge-attractor,* or a knowledge elicitation method [Bonnardel 93], which encourages and helps designers to articulate and input design knowledge into the system, and thereby partially overcomes knowledge acquisition problems.

At the Right Time. Intervention strategies determine when a system delivers knowledge. There are two extremes of intervention, *active* and *passive*. Active delivery mechanisms act like active agents by continuously monitoring user actions. Whenever the delivery mechanism identifies that the designers need information, such as when starting a design, making mistakes, or producing conflicts, the system automatically presents the designers with the knowledge relevant to the task at hand. Passive delivery mechanisms are explicitly invoked by users when they want to access some relevant information. Unlike information access mechanisms, with passive delivery mechanisms, designers do not need to form any queries or specific actions in order to represent their intention needs. The delivery mechanism autonomously searches for information relevant to the designers' task at hand, which is identified implicitly through designers' partial specification and construction.

Active delivery mechanisms, such as critics, are especially effective if the delivered information is related to identifying unsatisfactory conditions in the current design. Intervention immediately after a suboptimal or unsatisfactory action has occurred (immediate intervention strategy) has the advantage that the problem context is still active in the designers' mind and the designers still know how they arrived at the solution. Studies [Lemke 89; Carroll, McKendree 87] showed that passive critics were often not activated early enough in the design process to prevent designers from pursuing solutions known to be suboptimal. Often, subjects invoked the passive critiquing system only after they thought they had completed the design. By this time, the effort of repairing the situation was prohibitively expensive. In a subsequent study using the same design environment, an active critiquing strategy was shown to be more effective by detecting problematic situations early in the design process.

On the other hand, active delivery has the disadvantage that it may disrupt designers' cognitive processes and cause short-term memory loss. Designers then need to reconstruct the goal structure that existed before the intervention.

When building active delivery mechanisms, there is a problem of how to segment sequential tasks. When designers' tasks are not discrete, but are continuous and in real time, it is difficult for the system to differentiate a point at which the designers are done from an anomalous condition

[Silverman 92]. For example, in designing a kitchen, computing the total cost of design elements at the beginning of the design is absurd. The granularity of segmentation determines the time frequency of delivery. A finer grain of analyses may make the delivery more disruptive. A larger grain, on the other hand, would delay feedback past the cost-recoverable point, as discussed above [Carroll, McKendree 87].

The problem of intrusiveness is related to the discussion of how to deliver the right thing in the right style. That is, if delivered information is very useful and presented in a nondisruptive manner, it becomes less intrusive for designers. What is needed is a strategy that allows designers to control the intervention strategy of the delivery.

In the Right Style. Presenting information in the right style relates to the issue of which representations help designers to understand, modify, and apply delivered knowledge. Different situations need to view the same design knowledge in a different manner. People's ability to solve quantitative problems in real situations depends on (a) the knowledge they have thus acquired and (b) the ability to "transfer" that knowledge to their specific situation [Greeno 89]. The reasoning is situated in that it uses resources in the situation rather than computations involving symbols to arrive at conclusions. The type of representation that is most effective will depend upon the material to be presented and the way it is to be used [Norman 93]. Thus, depending upon the material and the task, some representations will be better than others.

Just delivering design knowledge from knowledge bases is not enough. It should be related to the designers' situation, and the rationale as to why the system derived the particular information should be available for designers. Knowledge delivery mechanisms in a design environment provide the information relevant to the current task to designers who have been involved in the task. They are situated in the current task; therefore, delivered design knowledge can be easily understood because it is close to their problem situation. Norman [1993] listed three requirements for good representations: (1) they must capture the important, critical features of the represented world, (2) they must themselves be in an appropriate format to enhance the person's access to the information they represent, and (3) they must support and simplify the computations needed to deduce implications or to determine regularities and structure within the representations.

Because the information delivered by KID, as discussed in the following chapters, is not complex, little effort has been paid in deciding appropriate representations. Different design domains such as delivering software objects, may require serious consideration of the appropriate representation required to support designers in understanding the delivered information [Redmiles 92].

Finally, the method to draw designers' attention to delivery is another issue. Wroblewski et al. [1991] have used the term *"advertising,"* which means drawing the users attention to the work materials rather than drawing attention to a separate notification window. Oppermann [1992] studied several presentation techniques to deliver information (to present suggestions for users) with the FLEXCEL system, which is an extension of adaptivity to EXCEL. The results show that

users were frustrated when suggestions interrupted their work (users were not allowed to continue the work unless they explicitly responded to the suggestion). The revised version of FLEXCEL notified designers that suggestions were made only with sound and visual icon and offered an adaptation tool bar, which allowed users to stock suggestions for later consideration. Oppermann concluded that the revised adaptive component enhanced users' appreciation of the adaptivity by providing them with control over the delivery.

3.5. Summary

In this chapter, I have discussed how computer systems should support designers based on the seeing-framing-seeing cycle while acknowledging the two intrinsic features of design: its ill-structured and open-ended nature. I claim that a design environment should be grounded on the human-computer cooperative problem-solving paradigm, so that the system augments rather than replaces the designer's skills. Two types of support mechanisms for information use, access and delivery, were identified, and challenges for effective information delivery mechanisms were discussed.

In the next chapter, I describe the multifaceted architecture for such design environments as a framework for my research. The framework provides a bridge from the theory that has been discussed to a system-building effort. The architecture provides: (1) tools for representing problem specification and solution construction, (2) knowledge bases that provide argumentation about design and a collection of previously built cases in a catalog, and (3) knowledge delivery mechanisms that support designers to locate the stored knowledge relevant to the task at hand. A prototype design environment named KID has been built based on the architecture.

Chapter 4

Framework: A Multifaceted Architecture for Domain-Oriented Design Environments

Through the discussions in Chapter 2 and Chapter 3, two requirements have been identified for computer-based design environments to support the *seeing-framing-seeing* cycle of design processes.

- Design environments need to provide tools that allow designers to explicitly represent a partial problem specification and a solution construction.
- Design environments need to provide delivery and access mechanisms that allow designers to exploit stored design knowledge relevant to the task at hand.

Figure 4-1 illustrates how a design environment that satisfies the above two requirements supports the seeing-framing-seeing cycle. In order to *see* a partial design that constitutes both a partial specification and a partial construction, designers may need information, such as design principles presented in a design textbook. In order to *frame* a partial design — that is, to develop a partial specification and construction following the reflection — designers may need information to make a new move, such as selecting which design component to use. During a cycle of these processes, designers may become able to articulate design knowledge that has not been stored in the system and to store the information in the system. Because both the development of a design and the use of knowledge bases take place in the same design environment, the system has more shared understanding, or information about the context that designers are currently engaged in.

During the last several years, several prototype systems of domain-oriented design environments [Fischer, McCall, Morch 89; Lemke, Fischer 90] have been developed and evaluated in order to achieve the above requirements. These different system-building efforts have led to development of a *multifaceted architecture*. The architecture provides (1) components for developing specification and construction; (2) two types of knowledge bases, an argumentation base and a catalog base; and (3) embedded knowledge delivery mechanisms.

In this chapter, I first briefly present a synopsis of the historical background of the multifaceted architecture. Then I describe components of the architecture and introduce the KID design environment, a kitchen floor plan design environment based on the architecture.

4.1. Historical Development of the Multifaceted Architecture

The multifaceted architecture has been developed through several prototype system-building efforts. The architecture was an integral response to the deficiencies that had been identified through testing these prototypes. The architecture is the integration of several paradigms, supporting design construction, design rationale, and reuse.

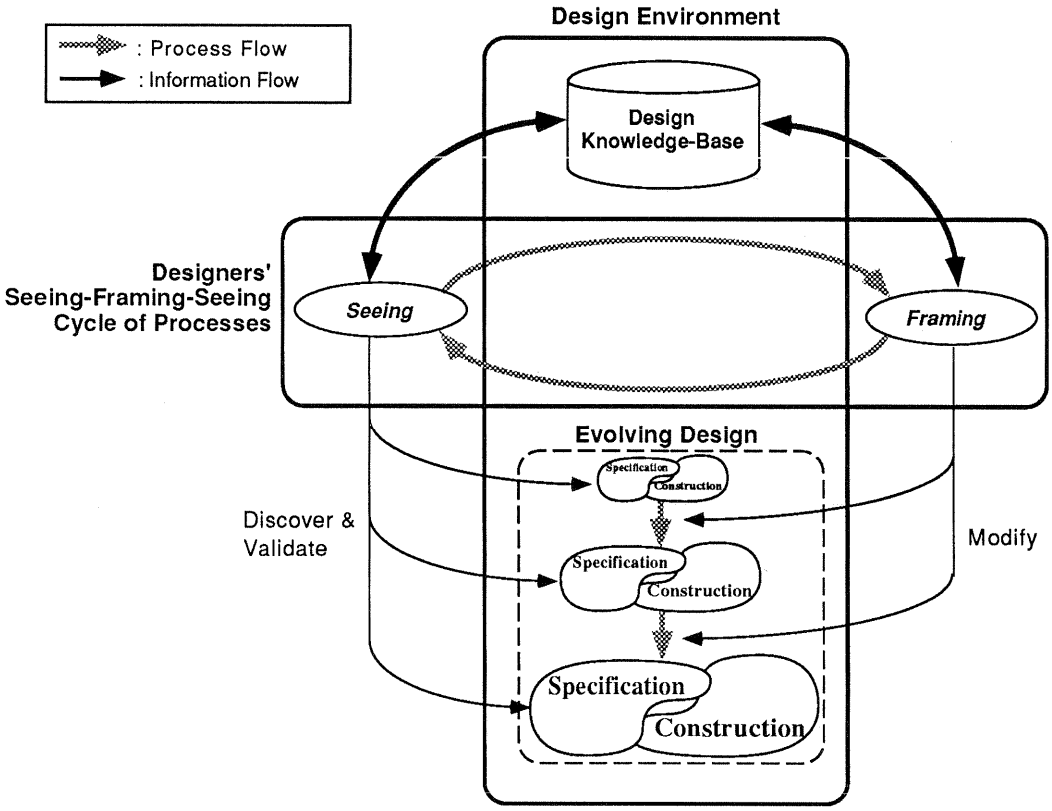


Figure 4-1: Design Process Model in Design Environments
Designers iterate *seeing* and *framing* while coevolving a partial problem specification and solution construction. Design knowledge is required to effectively support the cycle.

Human problem-domain communication. In 1987, Fischer [1987] started to develop the notion of a "construction kit" for supporting designers with computers. The construction kit provides domain concepts represented in object-oriented abstractions, which designers can manipulate on a computer display without any programming knowledge required. The construction kit was an attempt to provide *human-problem domain communication* [Fischer, Lemke 88], in which designers perceive design as communication with an application domain rather than as manipulation of symbols or lines on a computer screen. A domain-oriented system can support human problem-domain communication by storing and providing the important abstract operations and objects within the domain. Such an environment allows designers to design artifacts from application-oriented building blocks of various levels of abstractions according to the principles of the domain.

Adding a critics component. Some problems became apparent with the construction kit approach. Although the system provided domain abstractions to users, it had no knowledge about the quality of the design. In some sense, the construction kit was nothing more than plastic models, and the system was not cooperative in solving a design task. The construction kit helped designers in designing, but not in developing a *"good"* design. This led to the development of the notion of a "design environment." Lemke [1989] has developed FRAMER, a design environment for user

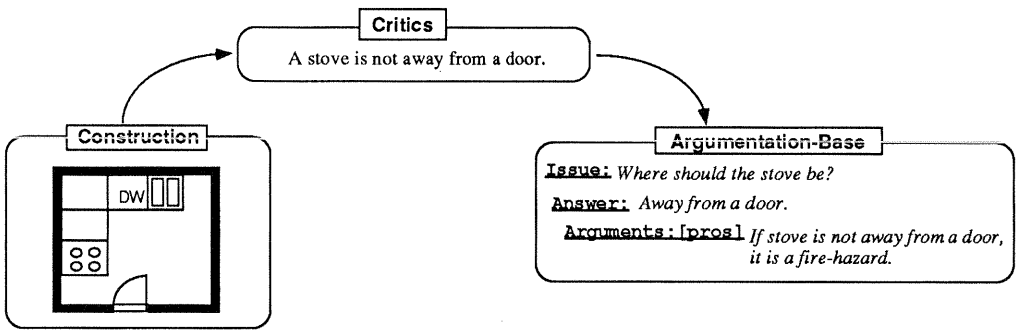


Figure 4-2: Components of the JANUS environment

A critiquing component in JANUS identifies a problematic situation in the construction in terms of a general design principle. A displayed critic message is linked to a corresponding argument in the argumentation base, which provides a further explanation of the principle.

---

interface design. FRAMER provides critics as a mechanism to give feedback about the design. About the same time, Morch developed another design environment for a kitchen floor plan design, CRACK [Fischer, Morch 88], which was later refined into JANUS-CONSTRUCTION as described below.

Integration of construction and argumentation. Testing of the two systems, FRAMER and CRACK, revealed that just having a critiquing mechanism is not enough. Users want to have explanations about the critique, which is often abstract and hard to contextualize in terms of their task at hand. JANUS [Fischer, McCall, Morch 89] was developed to integrate JANUS-CONSTRUCTION with JANUS-ARGUMENTATION, a hypermedia system that supports designers to discuss design alternatives. JANUS-ARGUMENTATION was based on PHI, which had been developed by McCall [McCall 91].

JANUS was a first attempt to support reflection-in-action by integrating a hypermedia system that records design rationale with a construction kit through critics. The system supported designers in performing the *seeing-drawing-seeing* cycle, in which the critics identify a potential breakdown in the partial construction, and a critic message that is fired is associated with a further explanation in the argumentation component (see Figure 4-2). Later, a catalog of floor plans was added to JANUS to support reuse. MODIFIER [Girgensohn 92] was implemented in addition to JANUS to allow designers to add and modify design objects in the system without programming.

JANUS, however, had shortcomings. The system supported only *drawing*. Designers had to *see* their partial construction in terms of the problem specification that was only in mind. The insufficient support for *framing* a partial problem and solution put another limitation to the system. By not knowing designers' intentions, the system can only provide generic design knowledge, not necessarily relevant to the designers' task at hand.

We have identified that the design environment needs to have a specification component, which allows designers to specify their goals and requirements for the design. The conclusion has led to

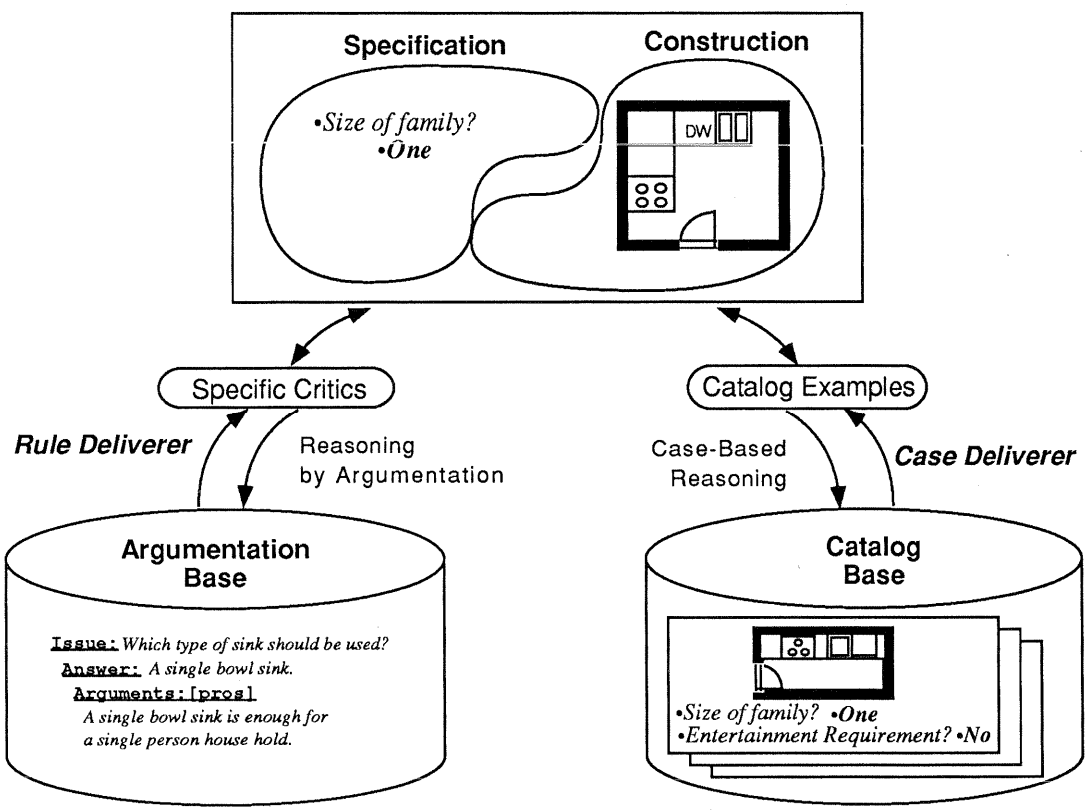


Figure 4-3: Components of the Multifaceted Architecture

KID is based on the multifaceted architecture, which consists of tools to produce a design and knowledge bases, allowing designers to develop designs from multiple perspectives. A specification component supports designers to articulate design goals, objectives, criteria, or constraints. A construction component supports designers to build designs in a solution form. An argumentation base provides design principles in the domain and design rationale based on the PHI structure. A catalog base provides case-based information, which is a set of previously constructed design, including both specifications and constructions.

A partial specification and construction represents designers' task at hand (represented with a rectangle). Specific critics identified by RULE-DELIVERER notify designers of possible violations of design principles in the construction in terms of the partial specification. The design principles are linked to arguments in the argumentation base. CASE-DELIVERER automatically orders the catalog examples according to the identified task at hand.

---

the development of a multifaceted architecture, a domain-independent architecture for a class of domain-oriented design environments. KID has been developed based on this architecture.

4.2. Components of Design Environments

Integrated design environments based on the multifaceted architecture are composed of the following components (Figure 4-3):

- A **construction component** is the principal medium for implementing design. It provides a palette of domain abstractions and supports the construction of artifacts using a direct manipulation style. A construction represents a concrete implementation of a design and reflects a designer's partial solution to the design task.

- A **specification component** allows a designer to describe objectives and required characteristics of the design at a high level of abstraction, and to assign weights of the importance to each specified item. The specifications are expected to be modified and augmented during the whole design process, rather than to be fully articulated before starting the design. A specification represents a designer's partial problem to the design task.
- An **argumentation base** captures the design rationale, which consists of articulated reasons that led to design decisions. Information fragments in the hypermedia issue base are based on an issue-based information system consisting of an issue, alternative answers, and associated arguments provided by the PHI (Procedural Hierarchy of Issues [McCall 91]) structure, and are linked according to what information serves to resolve an issue relevant to a partial construction.
- A **catalog base** provides a collection of produced design artifacts that have been accumulated through the use of the design environment. Designers can store both a completed construction (solution) and specification (problem) into the catalog base. Catalog examples can be used for reuse and case-based reasoning by illustrating the space of possible designs in the domain.

The architecture derives its essential value from the integration of its components and links between the components. Used in combination, each component augments the value of the others, forming a synergistic whole. Used individually, the components cannot effectively support designers in the *seeing-framing-seeing* cycle. Designers have to *see* the partial design in terms of both a partial specification and construction using design knowledge stored in the system. Links among the components of the architecture are supported by knowledge delivery mechanisms that deliver design knowledge from the argumentation base and the catalog base.

A knowledge delivery mechanism of the design environment consists of two parts. First, the system has to analyze a partial specification and a partial construction, and partially identify the designers' task at hand. ''The task at hand'' represents high-level design concerns of the current evolving artifact. Second, the system has to present knowledge or information to designers relevant to the task at hand. A partially identified task at hand through the first step can be used as a background query for information retrieval, and consequently the system can make the information space relevant to the task at hand. The knowledge delivery mechanisms are not independent subsystems but are embedded in other components.

4.3. KID: Design Environments for Kitchen Design

KID has been developed based on the multifaceted architecture. KID has three main subsystems. CONSTRUCTION supports designers to develop a design solution, KIDSPECIFICATION supports designers to articulate goals and requirements for the design task, and CATALOGEXPLORER supports designers to explore the catalog space. Because KID is based on JANUS, functionalities that existed in JANUS, including CRITICS [Fischer et al. 91a], which analyzes a partial construction and identifies violations of general design principles, and MODIFIER, which supports designers to modify design objects in the system without programming, are also available. The argumentative hypertext component in KID is integrated into KIDSPECIFICATION.

Table 4-1: A Summary of Subsystems of KID

	Functionality	Reference
KIDSPECIFICATION	• Specification of requirements • A view to the argumentation base • Suggestions • Access to ordered catalog examples	Chapter 6 Chapter 7
CONSTRUCTION	• Construction of a floor plan • CRITICS • Access to ordered catalog examples • MODIFIER: end-user modifiability	Chapter 7 Fischer, McCall, Morch [1989] Girgensohn [1992]
CATALOGEXPLORER	• Search of the catalog base • Access to ordered catalog examples • Retrieval of catalog examples by matching specification • Retrieval of catalog examples by matching construction	Chapter 7 Fischer, Nakakoji [1992]

In addition to these subsystems, KID provides two embedded knowledge delivery mechanisms, RULE-DELIVERER and CASE-DELIVERER. The RULE-DELIVERER mechanism determines which critic rules should be enabled as *specific critics*, which warn designers of potential problems in the current construction in terms of a partial specification. KID provides two types of critics: *generic* and *specific*. Generic critics reflect design knowledge that is applicable to all designs, such as accepted standards or regulations or domain knowledge based on physical principles. Specific critics reflect the designers' goals as specified in the specification component and apply only to the design situation currently under consideration. A specific critic enabled by RULE-DELIVERER is linked to the related specification item, and also to an argument in the argumentation base, which provides further explanation about why the critic rule is significant. CASE-DELIVERER presents design examples from the catalog base ordered in accordance with the current specification. Designers can access rationale underlying the ordering of the provided catalog examples, and the rationale is tied into the argumentation-base.

Chapter 6 provides descriptions of the design and design rationale of KIDSPECIFICATION and how it is integrated with the other components of KID. A system description of KID and the mechanisms of RULE-DELIVERER and CASE-DELIVERER are provided in Chapter 7. Table 4-1 provides a summary of the system's functionality and a reference guide.[1]

[1]A detailed description of the CONSTRUCTION system is found in Fischer, McCall, Morch [1989], and that of MODIFIER is found in Girgensohn [1992].

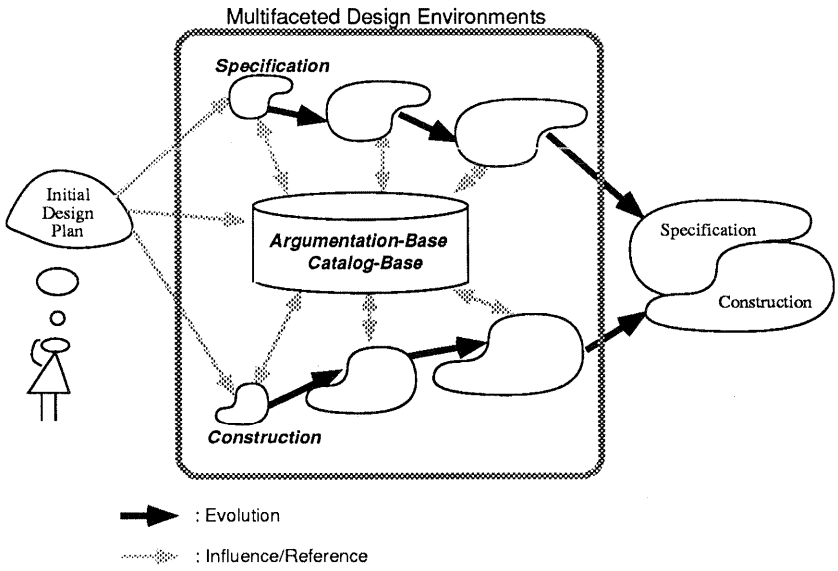


Figure 4-4: Coevolution of Specification and Construction

The coevolution of specification and construction in an environment based on the multifaceted architecture. A typical cycle of events in the environment includes: (1) designers create a partial specification or a partial construction, (2) the system provides them with information relevant to the partially articulated task at hand, (3) designers are able to refine their understanding of their partial design based on the *back talk* of the situation, and (4) designers modify a partial specification or construction. As designers go back and forth among the components, the problem space is narrowed and different facets of the artifact are refined. Design rationale identified during the design process can be accumulated into the argumentation base. A completed design artifact (consisting of specification and a construction) may be stored into the catalog base for later reuse.

4.4. Design Methods and the Multifaceted Architecture

The multifaceted architecture is not based on any particular design methodology. Rather, design environments based on the architecture provide an embedded design method for designers. By building prototype design environment systems, we can better understand design processes. The architecture for design environments and our understanding of the theory underlying design have been developed together. Without theory, we are unable to build a useful design environment. Without prototyping design environments based on the identified theory, we are unable to reflect on and refine the underlying theory. This coevolution of theory building and prototype building exactly reflects what we see as an appropriate design process model: *reflection-in-action.*

Figure 4-4 illustrates how designers develop their design using a design environment based on the multifaceted architecture. Designers create a partial specification or a partial construction from an often vague design goal. As the design environment provides feedback as knowledge delivery, designers develop the partial construction and specification gradually. Sometimes modification of a specification leads a designer directly to modify a construction, or vice versa. When designers become aware of the lack of information in the system and become able to articulate the new information, they can store the information into the knowledge bases, such as storing design rationale into the argumentation base. Instead of modifying the current design, designers may

replace the current construction or specification by reusable cases from the catalog base. A cycle ends when a designer commits the completion of the development, and the outcome is a specification and construction that conform to each other. The completed artifact, both the specification and the construction, can be stored back into the system's catalog base.

This model of coevolution of specification and construction is a design method inherently provided by the design environment architecture. This coevolution model and the *spiral model* of software design [Boehm 88] support iteration of the requirements specification and solution construction. The difference is that the spiral model places emphasis on discrete cycles of revision, remedy, and modification of design artifacts in terms of risk reduction, with a strict separation of cycles by reviews that are used to ensure the commitment to proceed to the next phase. In contrast, our model does not require any separation of the activities, but rather supports continuous growth of design, both in the specification and construction.

4.5. Summary

The multifaceted architecture is an instantiation of the approach that supports design by coping with its ill-structured and open-ended characteristics. Arias [1993] viewed design as an ill-defined problem-solving activity, involving decision making, reflection-in-action, flexible thinking, and creativity. Components of the multifaceted architecture support each of these facets of a design activity. *Decision making* is supported within the specification and the construction components. Designers can explicitly represent their decisions in terms of the domain context. *Reflection-in-action*, the main theme of this dissertation, is supported by having the explicit representation of the task at hand, and with the two knowledge delivery mechanisms. *Flexible thinking*, which supports reflection-in-action, is supported with the argumentation base. Designers can have access to pro and con arguments about a design decision made by other designers, which encourages designers to view a partial design from different points of view. Knowledge delivery mechanisms also support designers to think flexibly by triggering them to change their conceptual focus. *Creativity* is supported with RULE-DELIVERER and CASE-DELIVERER. As briefly discussed in Chapter 10, creativity is not just a mental activity [Boden 91], but is greatly enhanced by interacting — in the right way — with knowledge in the world [Norman 93]. By delivering to designers arguments and catalog examples that they may have never thought of before, the mechanism amplifies the designers' creativity by "bringing existing design concepts into unseen and even unthought, yet valuable ways of usage" [McLaughlin, Gero 89].

This chapter provided a description of a framework on which this research is based. The KID design environment has been developed as the multifaceted architecture has been evolved. KID is integrated by establishing many links among individual subsystems. Its knowledge representations are distributed among the components because of the historical background. This dissertation does not present KID as *the* answer to implementing the idea of the architecture, but as a prototype system, which demonstrates that the idea *can be* implemented on a computational substrate. By having a working prototype at hand, we can "see" what the real problem is, and we can gain an understanding of the problems of design.

Chapter 5

Scenario: How KID Empowers Designers

In this chapter, I present two scenarios: one using JANUS and the other using KID. These scenarios illustrate how a designer is more empowered using KID than JANUS; JANUS supports only the seeing-*drawing*-seeing cycle of processes, whereas KID provides support for the seeing-*framing*-seeing cycle of design processes.

After presenting each scenario, I provide discussions of issues addressed by each system, emphasizing what has been achieved by having a specification component in KID. A complete list of commands and their detailed specification of KID is provided in Appendix A.

In both scenarios, a novice kitchen designer, Jeff, wants to design a kitchen for his brother, Joe. When Jeff asked Joe what he wants for his kitchen, Joe said, *''I live by myself and I am left-handed. That's it. I do not care about the rest.''*

5.1. Scenario using JANUS

The scenario presented in this section illustrates how Jeff designs a kitchen for Joe using JANUS. With the CONSTRUCTION (Figure 5-1) system, Jeff starts browsing catalog examples in the *Catalog* window, but cannot decide which one he likes. So Jeff decides to design from scratch rather than to reuse a catalog example. After constructing a framework with walls, a window, and a door, he picks up a double-bowl sink in the *Palette* window and places it in the corner of the kitchen (see *Work Area* in Figure 5-1). A critic message appears in the *Messages* window, saying *''The sink is not in front of the window.''* Jeff does not understand why this is significant, and clicks the mouse on the message. This causes the system to bring him to the ARGUMENTATION system, which contains a detailed description of why a sink should be in front of the window (see Figure 5-2). Jeff wants to see a concrete example that satisfies this design principle. He clicks on the *Show Example* command in the menu, then the system provides him with an example from the catalog, *Lorry-Kitchen*, as shown in the top right corner of the Figure 5-2.

He looks at *Lorry-Kitchen*, and decides that he likes it. He goes back to CONSTRUCTION, scrolls through the *catalog* window until he finds *Lorry-Kitchen*, and replaces his partial kitchen design with *Lorry-Kitchen* in the work area. A critic fires, saying *''The refrigerator is not close to a stove.''* So he moves the refrigerator closer to the stove, leading to the design shown in Figure 5-3.

He selects the command *Critique All*, and the system responds with the message *''This kitchen satisfies the principles of good design.''* Jeff is happy with the result, and he decides that he has finished his design for Joe.

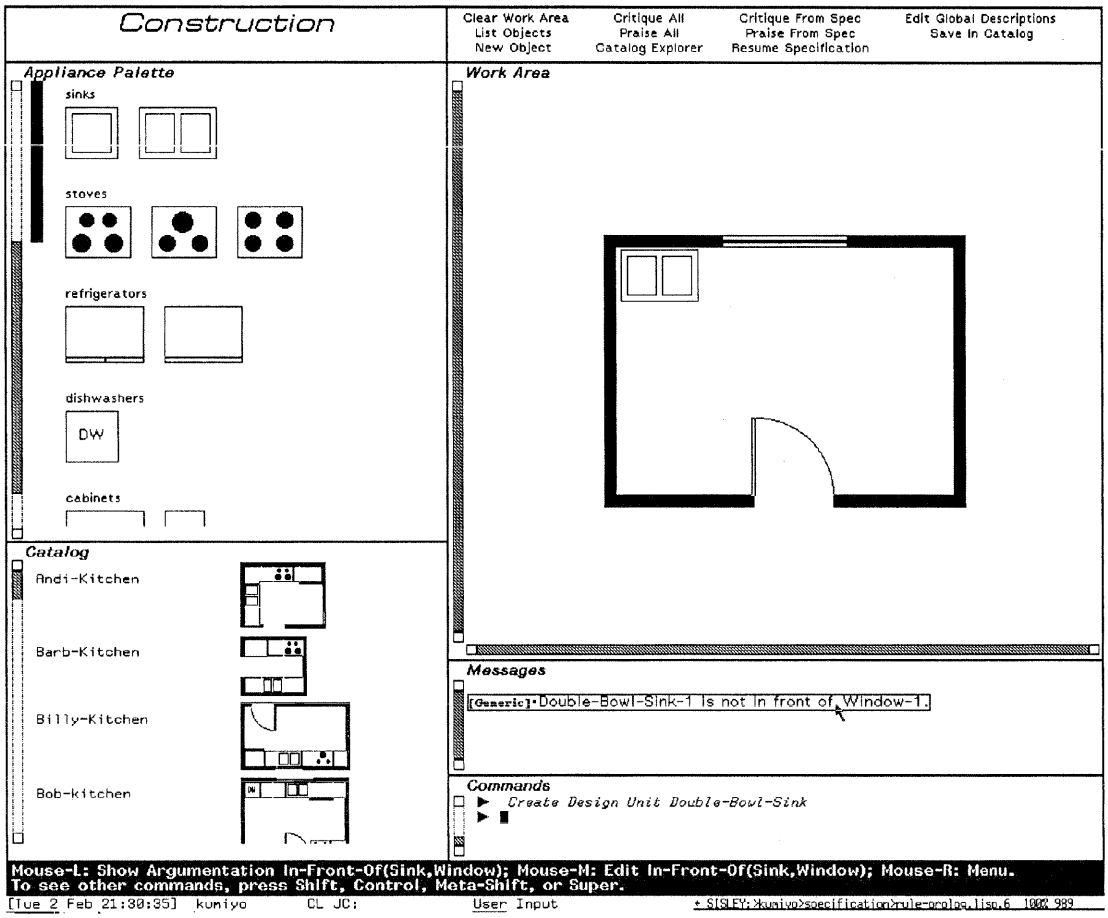


Figure 5-1: JANUS-CONSTRUCTION

Designers can construct a kitchen floor plan in *Work Area* by manipulating domain abstractions provided in the *Appliance Palette* window. Designers can also copy and edit a catalog example from the *Catalog* window. The *Messages* window displays a critique message that says the current construction in the work area is violating a design principle.

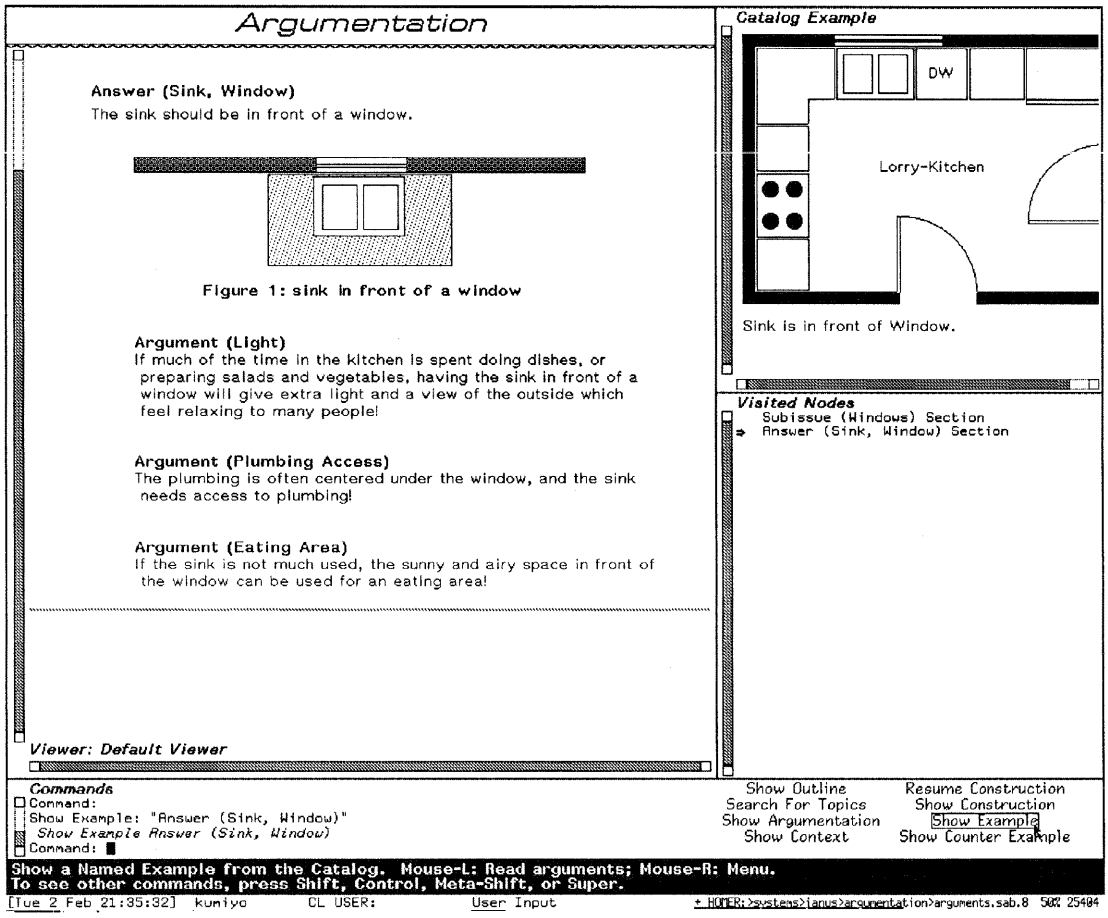


Figure 5-2: JANUS-ARGUMENTATION

Clicking on the critic message in Figure 5-1 brings Jeff to JANUS-ARGUMENTATION, in order to provide a further explanation of the critic. The *Show (Counter) Example* command displays a concrete (counter) example that illustrates the abstract argumentation.

5.2. Assessment of the Scenario using JANUS

As briefly described in Chapter 4, JANUS integrates the construction kit, an argumentation hypermedia system, a computational critiquing mechanism, and a catalog. As illustrated in the above scenario, JANUS:

- allows designers to construct a floor plan by picking up abstract design objects such as sinks, stoves, and refrigerators, provided in the *Appliance Palette*. Jeff could move, rotate, scale, and delete design units in the work area with the direct manipulation style.
- allows designers to reuse preconstructed design solutions from the catalog base. As presented in the scenario, Jeff was able to reuse *Lorry-Kitchen* and modify it to satisfy the critic.
- provides computational critics. Computational critics are mechanism that detect potential problematic features in a design and present users with a reasoned opinion about the quality of the design product. In the scenario, when Jeff first placed the sink on the corner of the kitchen, the critiquing mechanism detected the violation of a general design heuristic. The *Critique All* command allows designers to explicitly ask for the critics to examine the current design construction.
- allows designers to attend to further explanations about fired critics. In the scenario, Jeff's clicking on a displayed critic message automatically invokes the ARGUMENTATION system, which provides him with appropriate argumentation discussing issues in terms of placement of a sink.
- provides designers with concrete examples to illustrate the abstract argumentation. ARGUMENTATION presented Jeff with *Lorry-Kitchen* in order to illustrate the argument about location of a sink in terms of a window.

The scenario also highlights several shortcomings of JANUS:

- The system does not provide support for searching for specific catalog examples. Besides browsing, the only catalog access mechanisms provided by JANUS is the *Show Example* command in ARGUMENTATION that is available by attending to the argumentation base through fired critics. There would have been no support provided if critics did not fire, for example, before starting the construction. Also, Jeff's encounter with *Lorry-Kitchen*, which he happened to like, was rather serendipitous. No analytical reasoning was involved in determining that *Lorry-Kitchen* was a good example for Jeff's task, except that *Lorry-Kitchen* did not violate the design discipline about the location of a sink. The catalog might have tens of design examples that do not violate the discipline, and *Lorry-Kitchen* just happened to be one of them.
- Jeff's initial design requirement, Joe's being left-handed and a single person household, has not been reflected on in his design processes at all. Even if Joe, the client, had seven family members including five small children, Jeff could be finished with the same design without having any strong reasons.

In summary, JANUS supported designers in carrying out the seeing-drawing-seeing cycle of design processes. Because the problem specification (i.e., designing a kitchen for a left-handed, single household) was only in Jeff's mind and he did not have enough design knowledge to reflect on the construction in terms of the tacit specification, Jeff's ability in *seeing* was limited. Consequently,

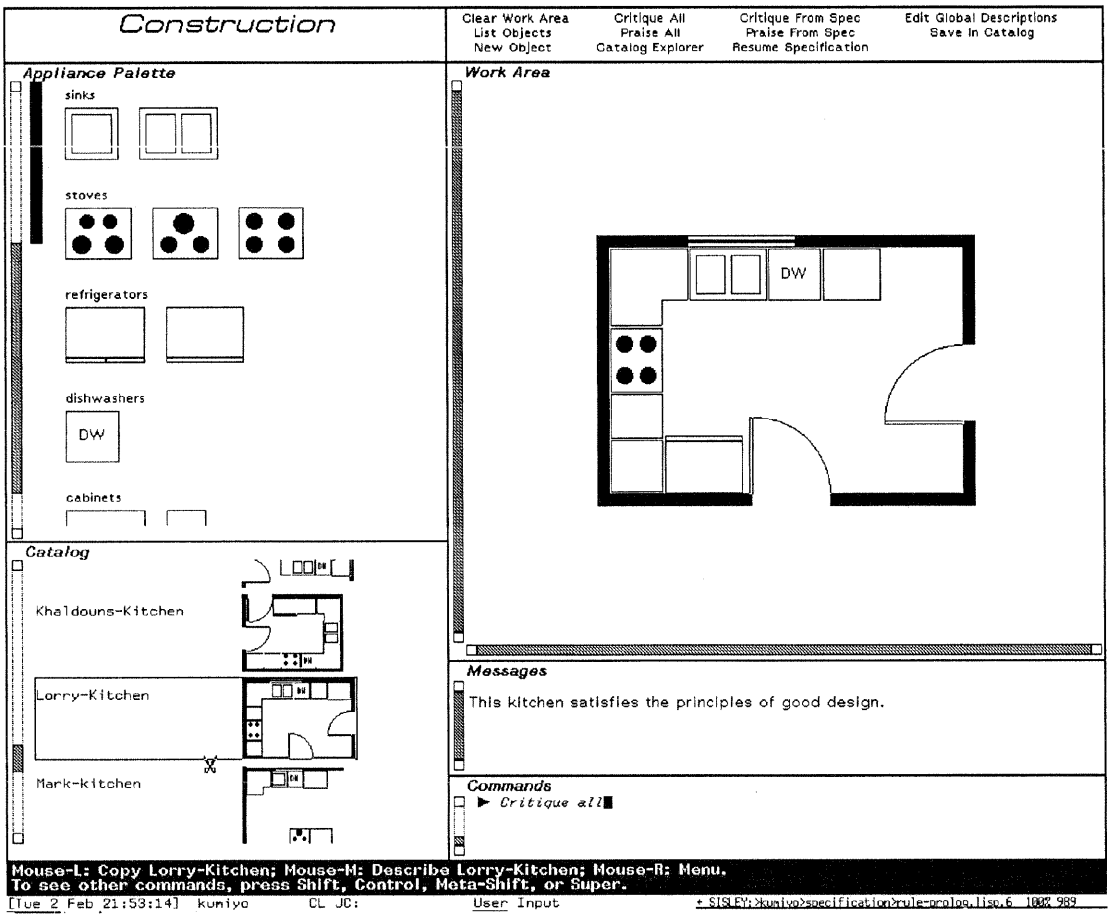


Figure 5-3: A Final Design in JANUS-CONSTRUCTION

The *Messages* window presents the information that Jeff's final design does not violate any critic rules that JANUS provides.

JANUS supported designers only at a *"generic"* design level. JANUS did not know Jeff's objective or intention in his design, and therefore could not support him except to provide general design principles.

The next scenario illustrates that the specification component provided in KID allows designers to specify their requirements and goals, making it possible for the system to support designers in reflecting on both the specification and construction, and exploring the argumentation base and the catalog base in terms of the specification.

5.3. Scenario Using KID

This scenario illustrates how KIDSPECIFICATION supports Jeff in evolving and reflecting on both his partial specification and his partial construction by providing (1) specific critics, which are made specific to his specified design requirements, and (2) catalog examples, which are ordered according to his specification. Three screen images of KIDSPECIFICATION (see the *Current Specifications* windows in Figures 5-4, 5-6, and 5-8) illustrate how Jeff gained his understanding of the design problem and evolved a partial specification, and two screen images of CONSTRUCTION (Figure 5-5 and Figure 5-9) illustrate how he evolved the construction. In what follows, short descriptions of the system's functionality are provided in footnotes.

Starting with KIDSPECIFICATION (see Figure 5-4), Jeff selects the *Start New Specification* command in the menu, and assigns a name to his design session as *Joe-kitchen*. Then, he starts browsing questions and answers provided in the *Questions* window. First, he answers that the client, Joe, is a single household by clicking on *"one"* to the question *"Size of the family?"* The answer appears in the *Current Specification* window on the top right of the screen. The next question *"Do both husband and wife work?"* seems irrelevant to Joe, so he skips it. The next question, *"How many cooks usually use the kitchen at once?"* is suggested to be answered as *"one"* because he has already answered that the size of family is one.[2] Jeff thinks the suggestion is reasonable, and confirms the suggested answer. Finally, Jeff specifies that the primary cook is *"Left handed."*

Looking at the current specification (see Figure 5-4), Jeff decides that he is ready to start constructing the floor plan, and chooses the *Resume Construction* command. In CONSTRUCTION (Figure 5-5), the system reorganizes the *Catalog* window and *Hill-Kitchen* is located at the top, suggesting that this example is most appropriate to his current specification.[3] Jeff looks at the example, and finds that he likes *Hill-Kitchen* except the single-bowl sink used. He copies the example into the work area and replaces the single-bowl sink with a double-bowl sink. Then, two critic messages are fired, one of which says *"A single bowl sink is not used"* (see the *Messages*

---

[2]KIDSPECIFICATION dynamically makes suggestions for designers about which answer to select.

[3]KID automatically orders catalog examples according to the partial specification.

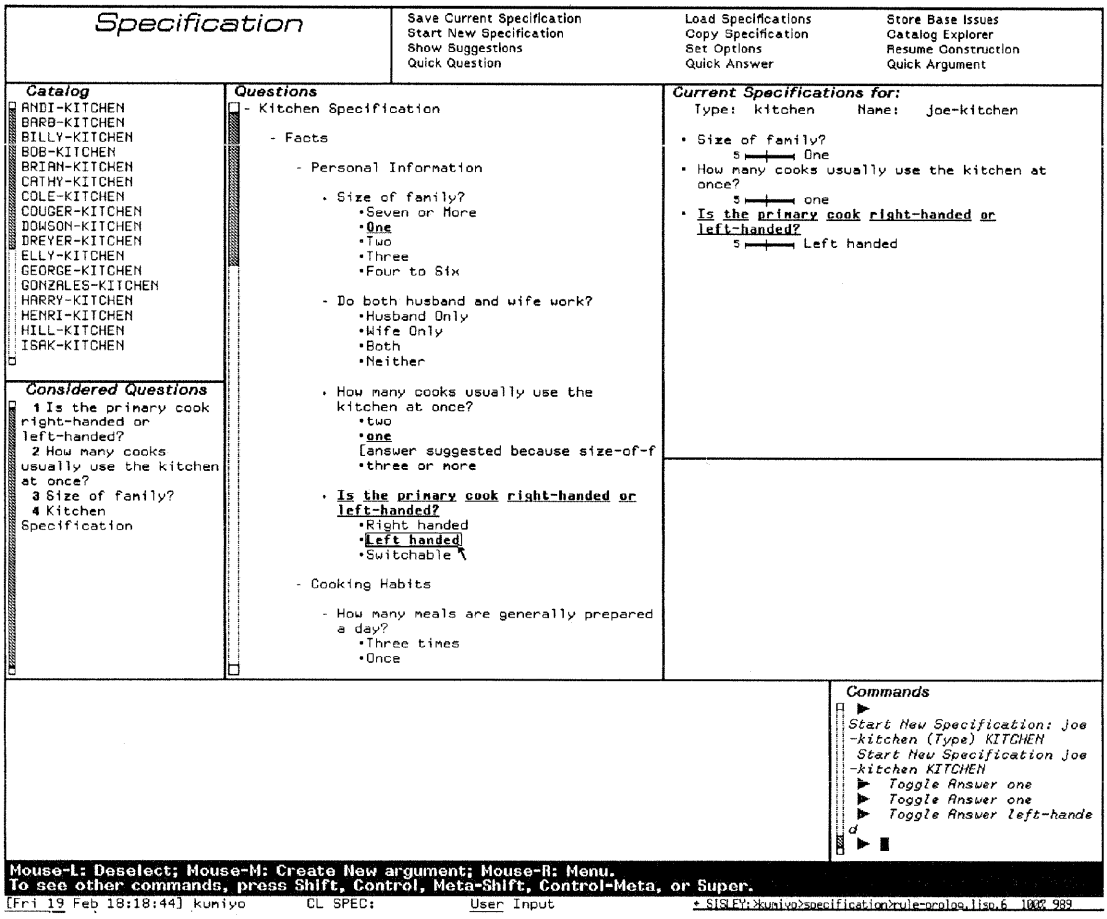


Figure 5-4: KIDSPECIFICATION

Designers can select answers presented in the *Questions* window. The summary of currently selected answers appears in the *Current Specification* window. Each answer is accompanied by a slider that allows designers to assign a weight representing the relative importance of the answer.

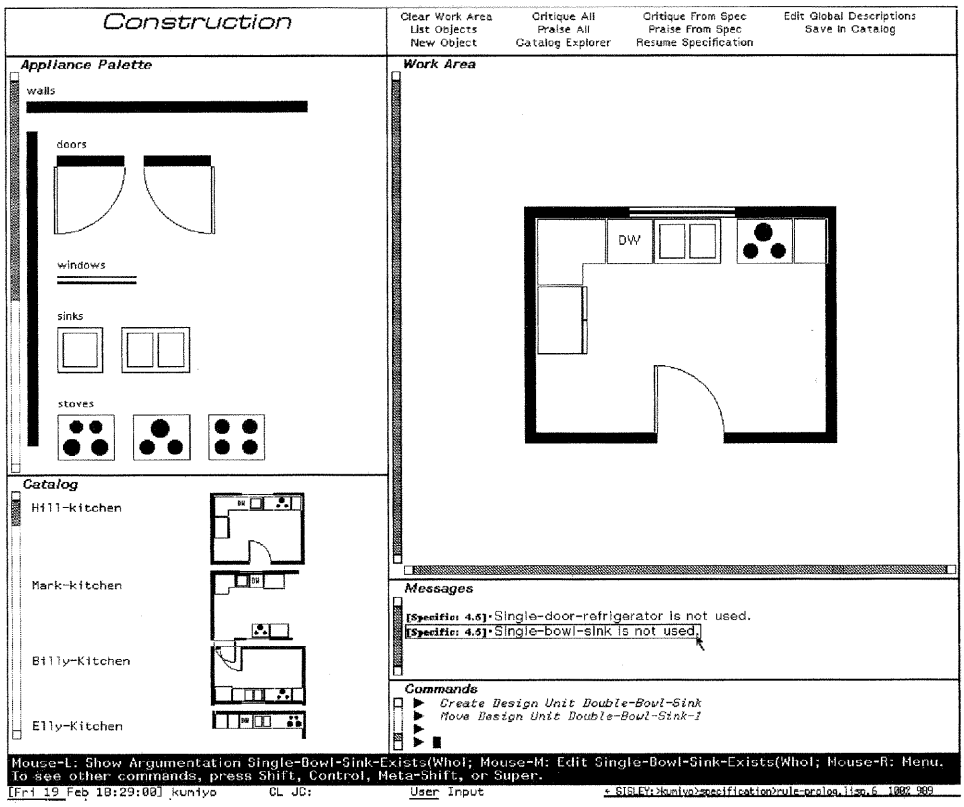


Figure 5-5: CONSTRUCTION

Catalog examples in the *Catalog* window are automatically ordered in accordance with the current specification (Figure 5-4). KID provides specific critics (see the *Messages* window), which are associated with numbers for their relative importance. A presented specific critic is linked to the related specification (see the *Suggested* window in Figure 5-6).

window in Figure 5-5).[4] Jeff wonders why this condition is necessary. When he selects *Show Rationale* from a menu associated with the fired critic, he is brought back to KIDSPECIFICATION, which explains that because the client is a single household, the kitchen does not need a double-bowl sink (see the *Suggested* window in Figure 5-6).[5]

Jeff wants to see further argumentation about the selection of this type of sink. He clicks on the suggested message in the *Suggested* window, then the argumentation about selection of types of sink shows up in the *Argumentation For* window.[6] He reads the arguments, and becomes aware that an alternative type, a double-bowl sink, has a pro argument that a double-bowl sink is appropriate if the client often entertains. Jeff remembers that Joe often entertains, so he clicks on this argument and he is presented with the question *"Entertainment requirement?"* to answer.[7] Jeff starts thinking that his brother Joe may have a roommate in the near future, and concludes that the entertainment requirement is a much more important concern than Joe's currently being a single household. Jeff assigns a weight of 10 as an importance factor for *"Entertainment requirement?: Yes"* by moving the attached slider, whereas he assigns 1 and 2 to the other requirements (see the *Current Specification for* window in Figure 5-6).[8]

Jeff decides to look into the catalog for better examples and selects the *Catalog Explorer* command in the menu. In CATALOGEXPLORER (Figure 5-7), the names of the catalog examples are ordered in the *Matching Designs* window differently from what was provided in CONSTRUCTION (see Figure 5-5) because Jeff reframed his specification by adding the entertainment requirement. Now *Elly-Kitchen* is at the top of the list. A floor plan, a specification, and a description of *Elly-Kitchen* is provided in the three windows in the middle.[9] In order to know why the system suggests that this example is most appropriate to his specification, Jeff invokes the *Show Delivery Rationale* command in the menu. Six conditions are listed as delivery rationale in the *Delivery Rationale* window.[10] Jeff wants to evaluate *Elly-Kitchen*, and selects the *Evaluate Example* command in the menu with the *"Praise the example from current specification"* option. The system describes that four out of the six conditions are satisfied in the example (see the *Praise of Elly-Kitchen* window in Figure 5-7).[11]Jeff wants to remember *Elly-Kitchen*, thinking that this may become an interesting example later, and puts it into the *Bookmark* window with the *Add to Bookmarks* command.

---

[4]Specific critics check the compliance of the construction to the specification.

[5]KIDSPECIFICATION explains why how the specific critic is related to the current specification.

[6]KIDSPECIFICATION provides an interface to the argumentation base.

[7]Some of arguments are linked to questions in KIDSPECIFICATION.

[8]KIDSPECIFICATION allows designers to assign weights to the partial specifications in order to represent the relative importance of each selected answer.

[9]In KID, a catalog example provides representations for both the construction and the specification.

[10]KID can explain how the system ordered the catalog examples.

[11]KID allows designers to evaluate a catalog example in terms of the current specification.

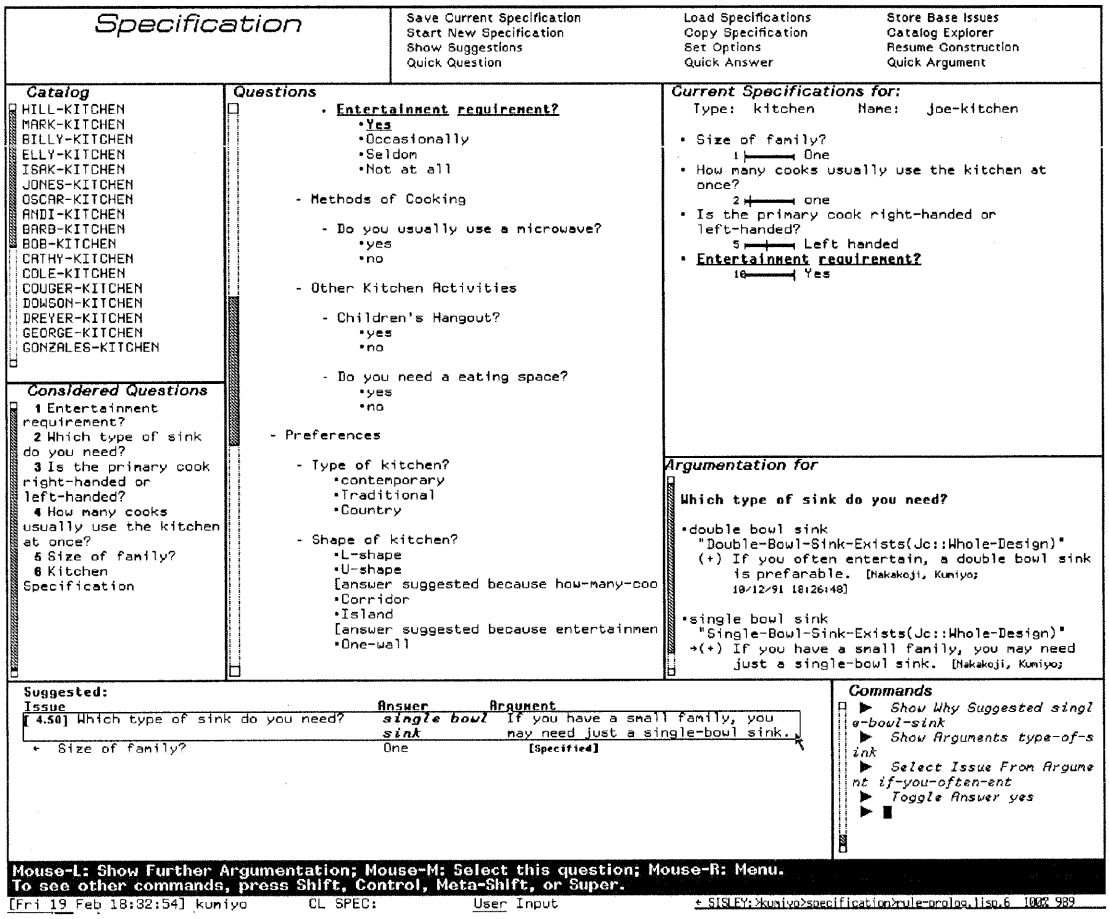


Figure 5-6: Reframing a Partial Specification in KIDSPECIFICATION

KIDSPECIFICATION provides an explanation about interdependency between a selected answer and a specific critic fired in CONSTRUCTION (Figure 5-5). The related argument (see the *Argumentation For* window) provides a further explanation about how the specific critic is related to one of the selected answers.

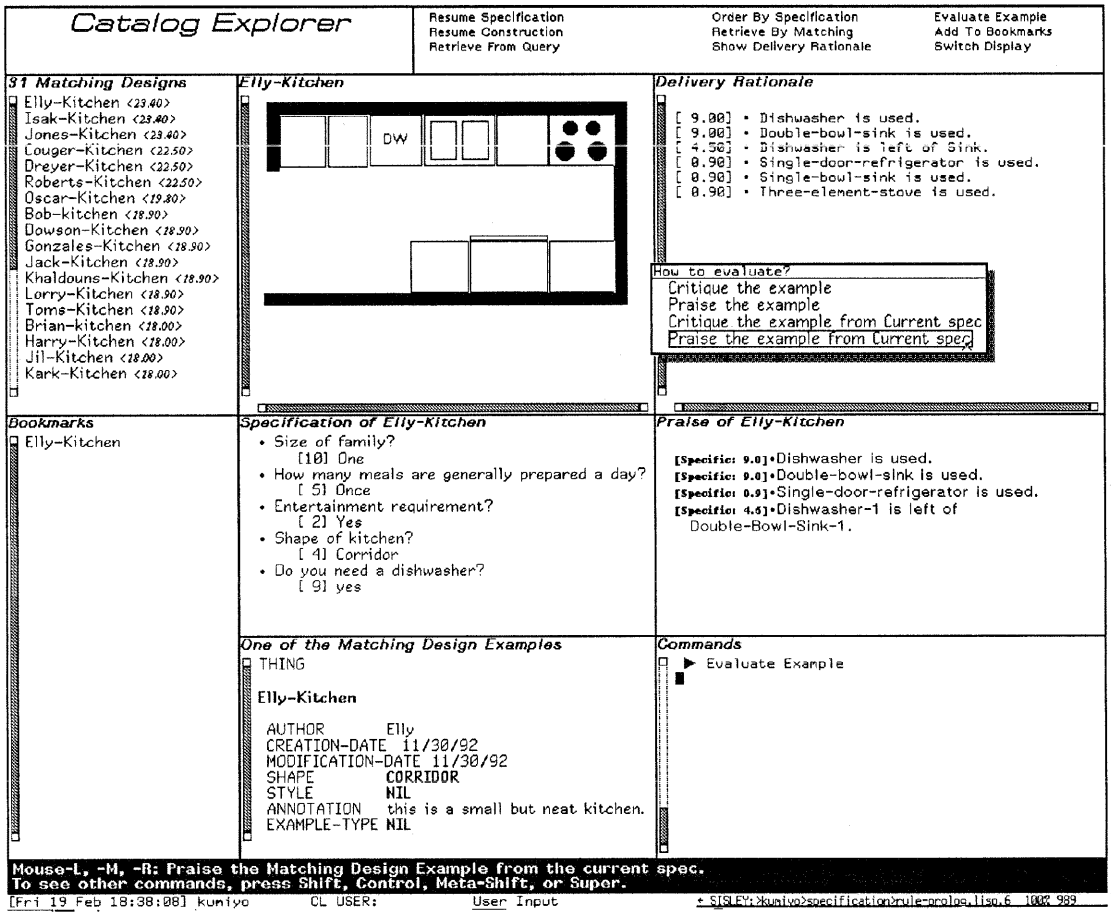


Figure 5-7: CATALOGEXPLORER

Designers can view the construction (a floor plan) and the specification (selected answers) of a catalog example in CATALOGEXPLORER (see the three windows in the middle column). Examples are ordered in accordance with the current specification (Figure 5-6) in the *Matching Designs* window, and the delivery rationale describes the ordering scheme. Designers can evaluate the presented example in terms of the current specification (see the *Praise* window).

Jeff reads the specification of *Elly-Kitchen*, and becomes aware that the specification has the *corridor shape* feature. He becomes interested in knowing which shape KID will recommend. He selects the *Resume Specification* command in the menu, and selects the *Show Suggestions* command in KIDSPECIFICATION (Figure 5-8). In the *Suggestions* window, the system lists suggestions made based on Jeff's current specification. The suggestions are ordered by numbers associated that indicate the relative importance of each suggestion. Jeff finds that the *"Island"* shape is suggested at the top with the importance value **9.00**. He realizes the *"!"* mark in the suggestion that notifies of the occurrence of conflicts with the island shape. He scrolls the window, and finds that *"U-shape"* is also suggested with the importance value **2.60**, because he specified that the design is for a single household, and that a U-shape kitchen provides a nice work flow for one cook.[12] Jeff sees further argumentation in the *Argumentation for* window about the selection of the shape of kitchen, and decides to select the island shape answer (see Figure 5-8).

Jeff decides to continue constructing his design and selects the *Resume Construction* command. In the *Catalog* window in CONSTRUCTION (Figure 5-9), catalog examples are reordered because he added the island-shape requirement to his specification. *Isak-Kitchen* at the second from the top gives Jeff an idea of having a stove in the island. He puts a stove in the middle of the kitchen with two counters. Jeff asks for critique with the *Critique All* command, and the system presents him with two messages about selection of the type of sink and refrigerator. Since the numbers assigned with the critics are small, indicating these violations are not very significant, Jeff decides that he has finished his design.[13]

5.4. Discussion

KID provides a specification component with which designers can specify their goals and intentions about the design task. This information enables KID to provide the designers with relevant design knowledge to their current task, which then supports designers in framing and reflecting on their specification and construction.

- KIDSPECIFICATION allows designers to explicitly specify the goals and objectives, gradually evolve the specification, and make trade-offs by assigning weights. In the scenario, first Jeff articulated that the task was to design a kitchen for a left-handed single-person household (Figure 5-4). Then he realized the entertainment requirement was more important than the requirement for a single household (Figure 5-6). Finally, he became aware of the shape feature of the kitchen (Figure 5-8). By having the explicit representation of the problem specification, designers can reflect on the partially framed problem.

- KIDSPECIFICATION guides designers on how to frame design specifications by making suggestions. For example, in the scenario, the system dynamically made a suggestion to Jeff to select *"How many cooks usually use the kitchen at once? — one"*, when he specified that this kitchen

---

[12]KIDSPECIFICATION can make ordered suggestions according to the current specification.

[13]Specific critics are weighted that shows the relative importance of the rules.

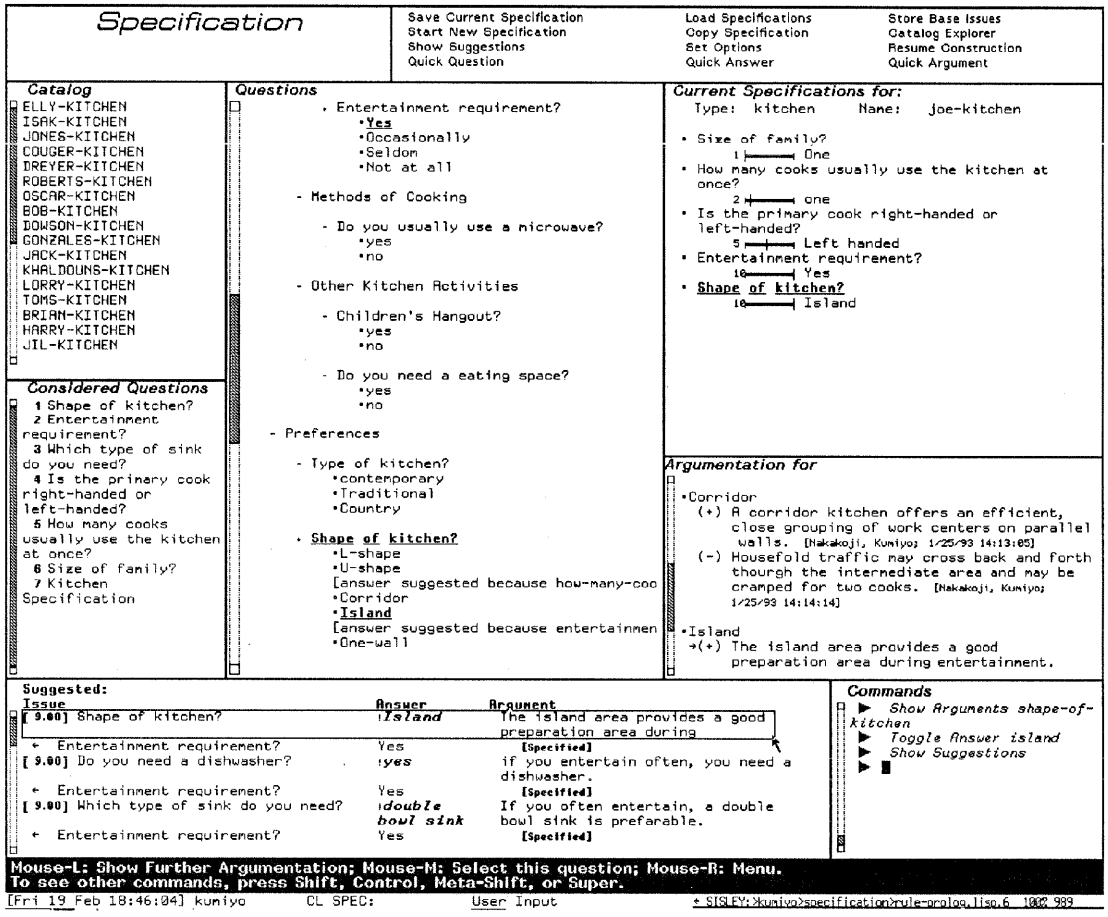


Figure 5-8: Examining Suggestions made by KIDSPECIFICATION

KIDSPECIFICATION makes suggestions (which answers to select) based on the current specification (currently selected answers). Suggestions are ordered according to the relative importance, with inference steps used to make each suggestion (see the *Suggested* window).

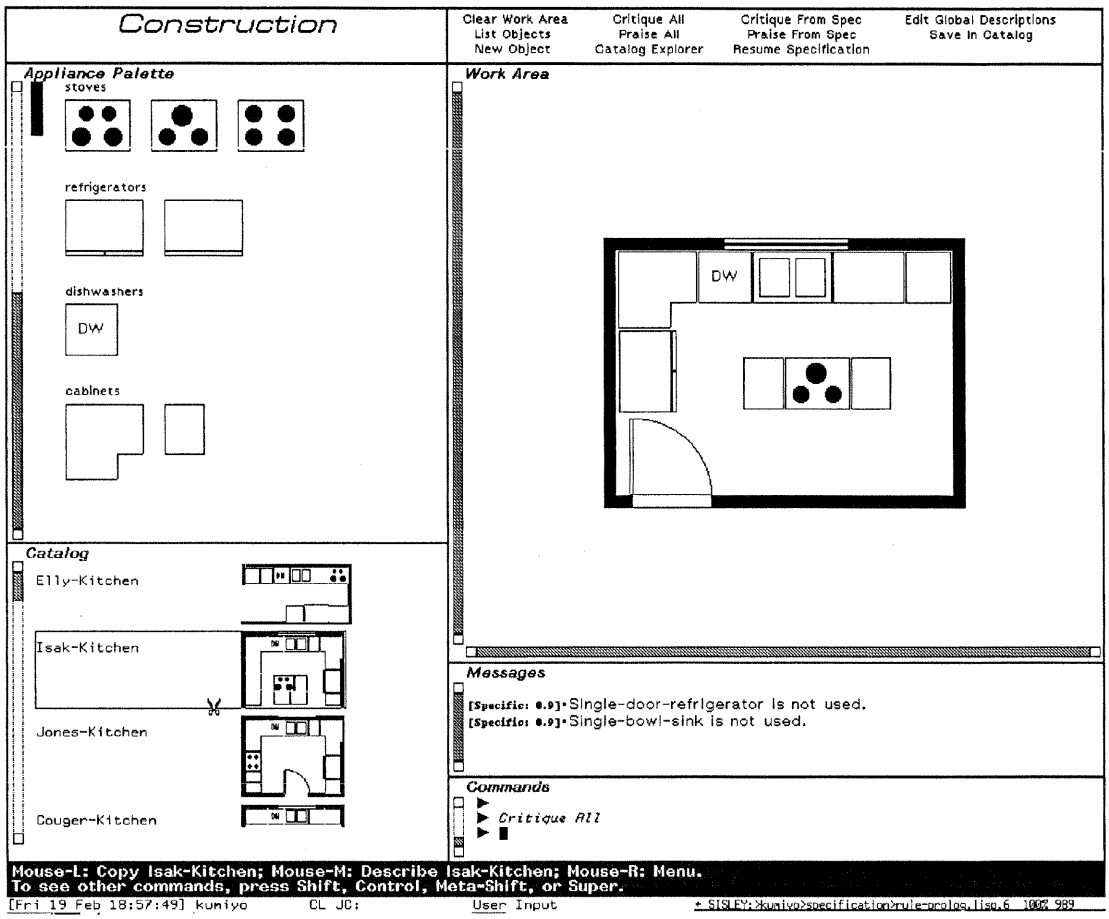


Figure 5-9: A Final Design in CONSTRUCTION in KID

Ordering of catalog examples presented in the *Catalog* window (and in the *Matching Design* window in CATALOGEXPLORER) is always updated according to the current specification.

was for a single household. KIDSPECIFICATION also made a list of suggestions for Jeff such as the island-shape of the kitchen. Suggestions are ordered according to the numbers that indicate the relative importance, thus, helping designers to understand which suggestions are more significant than others.

- KID allows designers to evaluate the design construction in terms of their problem specification. Some of critics are fired based on the current specification; i.e., specific critics. The fired specific critic messages are associated with further explanations, which have links to the specification and the argumentation. In the scenario, Jeff was first notified with a message that his current construction was potentially problematic because it did not have a single bowl sink. He was given a further description in KIDSPECIFICATION that because he specified that the kitchen was for a single household, the kitchen did not need a large sink. Designers can also evaluate a floor plan of a catalog example in terms of the current specification, as Jeff did with the *Evaluate Example* command (see Figure 5-7).

- KIDSPECIFICATION supports designers in finding the catalog examples relevant to the design task at hand. In the scenario, Jeff was provided with *Hill-Kitchen* as most appropriate to his specification before he started to construct a floor plan. As he developed the specification, catalog examples were displayed in different order, always presenting the most appropriate design example at the top of the list. KID also provides rationale underlying its ordering. With the *Show Delivery Rationale* command in CATALOGEXPLORER, Jeff learned why *Elly-Kitchen* was judged to be most relevant to his specification.

KID increases the chance that designers will encounter useful design knowledge. Design knowledge stored in KID is integrated; different types of design knowledge are linked together. In JANUS, a fired critic was associated with the argumentation, which was linked to a catalog example. In addition, KID provides links to a partial specification. Examples in the catalog base of KID include both problem specification and construction, and designers can reuse either or both of them. Attached specification supports designers to better understand the construction of an example, and may provide designers with new ideas. In the scenario, Jeff became aware of the existence of the shape feature from the specification of *Elly-Kitchen*, which led him to refine his problem specification.

KID supports designers in the cycle of seeing-framing-seeing. Specific critics that check the compliance of the current construction to the current specification support designers in reflecting on the partial specification and construction. By providing related knowledge in the argumentation base, KID supports designers in framing their problem specification and solution construction. In the scenario, when a critic fired about the type of sink in terms of Jeff's specification that a kitchen is for a single household, Jeff modified his problem specification by adding the entertainment requirement instead of modifying his design construction.

In summary, KIDSPECIFICATION helped Jeff to understand his design problem better. Using the information presented by Jeff in KIDSPECIFICATION, KID could deliver more timely and useful design information to him. Using the delivered design knowledge, Jeff could modify the specification and construction, or could further explore the related information stored in the system while

gaining new design knowledge. Jeff could gradually frame the problem specification and solution construction, while making more use of the stored knowledge in KID with the help of the system's knowledge delivery mechanisms.

5.5. Summary

The two scenarios illustrate how KID has been evolved from JANUS by having a specification component and knowledge delivery mechanisms that became more effective afterward. The next chapter provides theoretical descriptions of a specification, and describe its role, design, implementation, and underlying mechanisms that integrate KIDSPECIFICATION with the other components of KID.

Chapter 6

Specification Component: Its Meaning, Role, Design and Mechanism

The main theme of the dissertation is to explore the role of the specification component in knowledge-based design environments. The scenario presented in the previous chapter illustrated how KIDSPECIFICATION supports designers in the cycle of seeing-framing-seeing. KIDSPECIFICATION allows designers to explicitly articulate a problem specification, and the information provided in KIDSPECIFICATION can be used by the system to deliver design knowledge relevant to the task at hand.

In this dissertation, I use the term "specification" to refer to representations of a statement of goals, objectives, criteria, or constraints for a design task. Design is a purposeful action in constrained situations. Designers always have some purposes or goals in mind, and at the same time, they have to deal with both explicit and implicit constraints that limit and guide them in design spaces such as a limited amount of resources, social and economic constraints, and the limitations of available technology. Designers are required to achieve their purposes while modifying, adding, relaxing or hardening given constraints. As discussed in Chapter 2, the problem is that these purposes and constraints cannot be completely articulated before starting to construct a solution, but emerge during the development of the solution.

A specification component in design environments provides a representation that captures the information about the purposes and constraints that designers were able to articulate.

A representation of the specification can be at any level of abstraction. For example, in the domain of kitchen design, a specification given by a client may include statements such as:

1. I need a kitchen that is good for entertaining.
2. I am left-handed.
3. I want to have a dishwasher.

The first specification states a goal. The second statement represents the situation of the environment. The third specification states a concrete requirement in terms of the solution structure.

In this chapter, I first discuss the conceptual description and meanings of "specification." Next, I articulate issues and challenges in designing a representation for the specification, and discuss the role that a specification component can play in the context of a design environment. Then, I describe the design of KIDSPECIFICATION, a specification component of KID. Specification-linking rules are used to integrate KIDSPECIFICATION with the other components of KID.

6.1. What is Specification?

Habraken [1987] described design from a social point of view: *design is a proposition and suggestion, aimed at the agreement of clients and designers.* Habraken stated that difference between design and other creative activities is that in design the agreement of concerned humans is only assumed. A specification provides a representation for a partial acknowledgment of the clients about suggestions made, and a partial understanding of the designers about the requirements and constraints that the clients embody.

One view of a specification is *a list of conditions of satisfaction* for design. These conditions are not necessarily stated formally, but rather the statements are informal, ambiguous, and inherently contradictory, and can never be complete. Although McLaughlin [1992] referred to Heidegger to argue that designers are not guided by having descriptions of a desired state of affairs in mind, my claim is that designers always have some purposes in mind. The purposes are not stated in a mathematically rigorous manner, but in more vague and ambiguous ways. The approach taken in my dissertation is that the content of the specification and the representation of the specification will evolve as design progresses, and the specification must allow designers to represent their purposes informally.

One way of representing design consists of function, structure, and behavior [Gero 90]. Structure is the representation that a construction embodies in the multifaceted architecture. Behavior emerges when the designed artifact is actually implemented. Function is defined in terms of the structure, the behavior, and other social, cultural, and economical factors.

A specification is a statement of required function, desired structure, and expected behavior. In the representation of a design construction, structure is represented as a set of *surface* features. Function and behavior of the design are *hidden*, not visible to designers without design knowledge. Behavior can be only inferred from the surface structures. Function can be inferred directly from the surface structures, or from the inferred behavior [Gero 90].

McLaughlin [1992] described four views for understanding the function of design: as goals, as force, as effects, and as involvement. With the function-as-goals view, the function of design can be understood in terms of performance, such as acceptance tests recommended by Draper and Norman in user interface design [Draper, Norman 84]. With the function-as-force view, the function of design can be understood in terms of the needs given from outside environments that lead to the current design. With the function-as-effects view, the function can be understood in terms of the effects that the design will cause. Finally, with the function-as-involvement view, the function of design cannot be understood until the design is immersed in the practices of its users.

It is impossible to provide a single view for understanding the function of design. At different levels of abstraction, and at different phases of design, designers need to view design from different points of view. Sometimes they may need to think of the function of design as a goal, and sometimes they may consider the function as an effect. When viewing function of design as involve-

ment, it is impossible to provide any a priori description of function because it is impossible to articulate all the usage situations [Winograd, Flores 86].

6.2. Design of a Representation for Specification

The above discussions led to identify requirements for representations of a specification. A representation for the specification needs to be:

- flexible enough so that designers can specify functional, behavioral, and structural requirements at various levels of abstraction, and
- manipulable by a design environment so that the system can provide dependencies among the specification and the construction.

Being able to state the problem informally is an essential strategy for dealing with the complexity of design problems. Features such as abbreviation, ambiguity, poor ordering, incompleteness, contradiction, and inaccuracy are frequently observed in human-human cooperative problem solving [Rich, Waters 86]. Designers must be allowed to specify the function of design in their situation model, for example, *"an efficient kitchen."* At the same time, design knowledge must be provided to support designers in mapping the abstract requirement to concrete structural features, such as *"a kitchen that has a work triangle less than 20 feet."*

Viewing design as an argumentative process [Rittel 84; Fischer, McCall, Morch 89] is an approach for coping with the above requirements. Designers continuously argue among themselves, with clients, or with the material that they are dealing with, in order to gain an understanding of the design problem from different points of view. In order to describe the function, therefore, the statements should be ready to be argued. During the course of argumentation, designers accumulate design decisions. These design decisions may vary at structural, behavioral, or functional levels.

As shown in Chapter 5, the specification component of KID, KIDSPECIFICATION, is designed and implemented according to the forms of questionnaires that professional kitchen designers use to elicit clients' requirements. Analyses of the questionnaires reveal that questions cover different levels of functional statements as well as structures and behavior. For example, the question *"Size of family?"* represents a function as force; the question *"Are you interested in resale value?"* represents a function as effect; and the question *"Do you need a dishwasher?"* represents a structural requirement.

A specification component in a design environment can go beyond static paper-pencil representations by providing dynamic management of interdependency among specifications and constructions, and by providing the related design knowledge from the system's knowledge bases. Functions, behavior, and structures are all interrelated. Sometimes, conflicts occur at a hidden-feature level, for example, among two functional statements. Sometimes function is derived as a by-product without being articulated [Schoen 92; McLaughlin 92]. Using design knowledge, designers are able to deal with interdependency among the specifications. KIDSPECIFICATION has to be able to support designers to deal with these interdependencies.

KIDSPECIFICATION is implemented as a hypertext interface for the argumentation base of KID, which is based on the PHI (Procedural Hierarchical Issue) structure [McCall 91]. PHI is a family of IBIS, whose primary structure is a network of nodes consisting of an issue, alternative answers to the issue, and arguments for the answers. The structure is used to record design rationale, processes and reasonings of decision making in design tasks. Issues and alternative answers raised in the course of design represent a partially framed problem and solution. Domain-orientation of the design environment makes a large part of the design rationale appear in a design task applicable to other design tasks. In other words, a large part of the same set of issues must be considered again and again in many design problems in the domain. In this regard, articulating positions to alternative answers constitutes a problem specification.

The KIDSPECIFICATION system provides prestored questions (issues) and associated optional answers. Designers can select what they think is the concern of their current design task. Designers can assign weights to the selected answers in order to represent the intensity over the selected answers and prioritize them. If designers do not find prestored alternatives to express the position, they can add or modify information in the underlying argumentation base. Thus, designers are not only in a listener's mode, with which the system asks questions and designers answer, but can also play speakers' roles. For the purpose of making the system look more user-friendly, I used the term ''question'' in the user interface, which is interchangeable with the term ''issue'' throughout this dissertation.

Designing KIDSPECIFICATION as an interface for the argumentation base satisfies the two prerequisites listed above. First, issues and answers in the argumentation base can be at any levels of abstraction. With KIDSPECIFICATION, designers can specify anything they want, from a very abstract functional requirement to concrete features that are aspects of design construction. The underlying argumentation base is ''semi-structured'' in a sense that designers can fill in as much or as little information in different fields as they desire, and the information in a field can be any type, such as natural language text [Lai et al. 88].

Second, *serve relationships* defined over the PHI structure [McCall 91] of the argumentation base can be used to represent interdependencies among design decisions (selection of answers). A serve relationship of PHI means that how one issue is answered determines how another issue should be answered. Issues and answers in KIDSPECIFICATION are linked to one another [Fischer et al. 91b]. Arguments associated with answers help designers decide whether to select them.

To make the content of specification a shared understanding, the environments need to be able to manage the interdependencies among the selected answers. Machine understanding of objects in KIDSPECIFICATION is provided in *intension* rather than in *extension* [Alexander 64]. The meaning of objects is defined in terms of relationships to other existing objects, not defined with mathematical formula nor exhaustive enumeration of their elements.

The serve relationships mentioned above represent these relationships. Evidence shows that sup-

port of such interrelationships among selected answers and structures in a form of construction is the main constituent of design knowledge [Steier 90]. Research by Bonnardel [1991] shows that novice designers cannot map high level functional description to constraints in the construction level, whereas experts can.

As previously mentioned, PHI is a variation of the IBIS (Issue-Based Information System) [Kunz, Rittel 70] structure. Although application of computer technologies to support IBIS has been investigated for quite a while in a number of variations [McCall et al. 90; Conklin, Begeman 88; Lee 90; MacLean, Young, Moran 89], little effort has been made to provide computability to deriving relationships in IBIS. In most of approaches, relationships among nodes are supported depending only on human judgment. In order to fully support the IBIS paradigm with computer systems, both human judgment and computability of relationships among nodes must be supported.

In KID, the serve relationships are represented by *specification-linking rules*. The rules are automatically derived from the content of arguments that have been stored by designers; thus the approach integrates human judgment and computability. The specification-linking rules play the main role of integrating the specification, construction, and knowledge bases of the environment. Derivation of the rules is further described in the following sections.

6.3. Mechanism: Specification-linking Rules

The specification-linking rules form a decision network of interdependencies among issue-answer pairs. Because the issue-answer pairs vary from abstract functional specifications (e.g., *''Entertainment Requirement?: Yes''*) to concrete structural requirements (e.g., *''Type of sink?: Double bowl sink''*), specification-linking rules provide associations among functional, behavioral, and structural specifications.

The specification-linking rules support designers to become aware of the *hidden features* of design. A *hidden feature* of design is defined as a characteristic of a design that is not apparent in the construction, such as a functional or behavioral statement. A structural feature that is explicitly represented in the construction is a *surface feature* of design. For example, one of the specification-linking rules used in the scenario presented in Chapter 5 stated that if you entertain often, you need a double bowl sink. *''Having a dishwasher''* is a surface feature of a kitchen design if the construction of the design has a dishwasher in it. Using the rule, a hidden feature, *''good for entertaining,''* is inferred from the construction of the design.

In this section, I first describe the specification-linking rules in more detail. The rules are automatically derived from the content of the argumentation base. How these rules are derived, used, synthesized, and presented are then discussed.

6.3.1. What Are Specification-linking Rules?

A specification-linking rule represents a computable interdependency between two question-answer pairs. Figure 6-1 shows an example of the specification-linking rule, *"Size-of-family=one → Type-of-sink=Single-bowl-sink."* The rule means that there is a dependency between a specification about the number of members in a household and the type of sink to be used in the kitchen design. As shown in Figure 6-1, the rule is derived from an argument in the argumentation base, stating that if you are a single-person household, you do not need a double-bowl sink.

The rules are by no means logical rules; they are *rules-of-thumb* [Fish 89], and therefore disputable. Therefore, the rules must not be used as production rules to produce a solution based on a given problem specification. The rules provide heuristics, and this type of information plays an important role as design knowledge [Bonnardel 91]. The rules support designers in identifying the hidden features of a design. With the above example rule, a designer can discover that a kitchen design that has a single-bowl-sink has a feature *"good for a single-person household."*

The rules are derived from the argumentation base in KID. The argumentation base provides a collection of design rationale that has been accumulated through design experiences. Design experiences occur both logically and temporally prior to the rule derived from them [Schoen 92]. Automatically deriving rules as a reciprocal transformation of previous design rationale addresses the problem of knowledge acquisition. The problem of knowledge acquisition is that in order to build a knowledge base, knowledge engineers have to interview domain experts to elicit rules about the normative cues and about the specific errors experts and nonexperts frequently commit. However, this is very time- and cost-consuming, and often experts cannot articulate rules in the first place [Bonnardel 93; Silverman 92].

The specification-linking rules are transitive; for example, if you are a single-person household, you do not need a large refrigerator, and if you do not need a large refrigerator, you do not need a large electric capacity. Therefore, if you are a single-person household, you do not need a large electric capacity. The requirement that a question-answer pair represents can vary from a hidden feature requirement, which is abstract and functional or behavioral, to a surface feature requirement, which is concrete and structural. Therefore, the specification-linking rules provide mapping among functional, behavioral, and structural features [Steier 90], and thus, the rules support designers to map abstract criteria to concrete structural constraints.

The rules increase the designers' awareness of conflicts among hidden feature specifications. When mapping hidden feature specifications to surface feature constraints, conflicts may become apparent among the surface feature constraints. For example, in the scenario, two abstract level specifications, *"Size of family?: One"* and *"Entertainment Requirement?: Yes"* cause a conflict about the type of sink: being a small household suggests a single-bowl sink, and the entertainment requirement suggests a double-bowl sink.

Finally, because the rules map high-level functional or behavioral specifications to low-level struc-

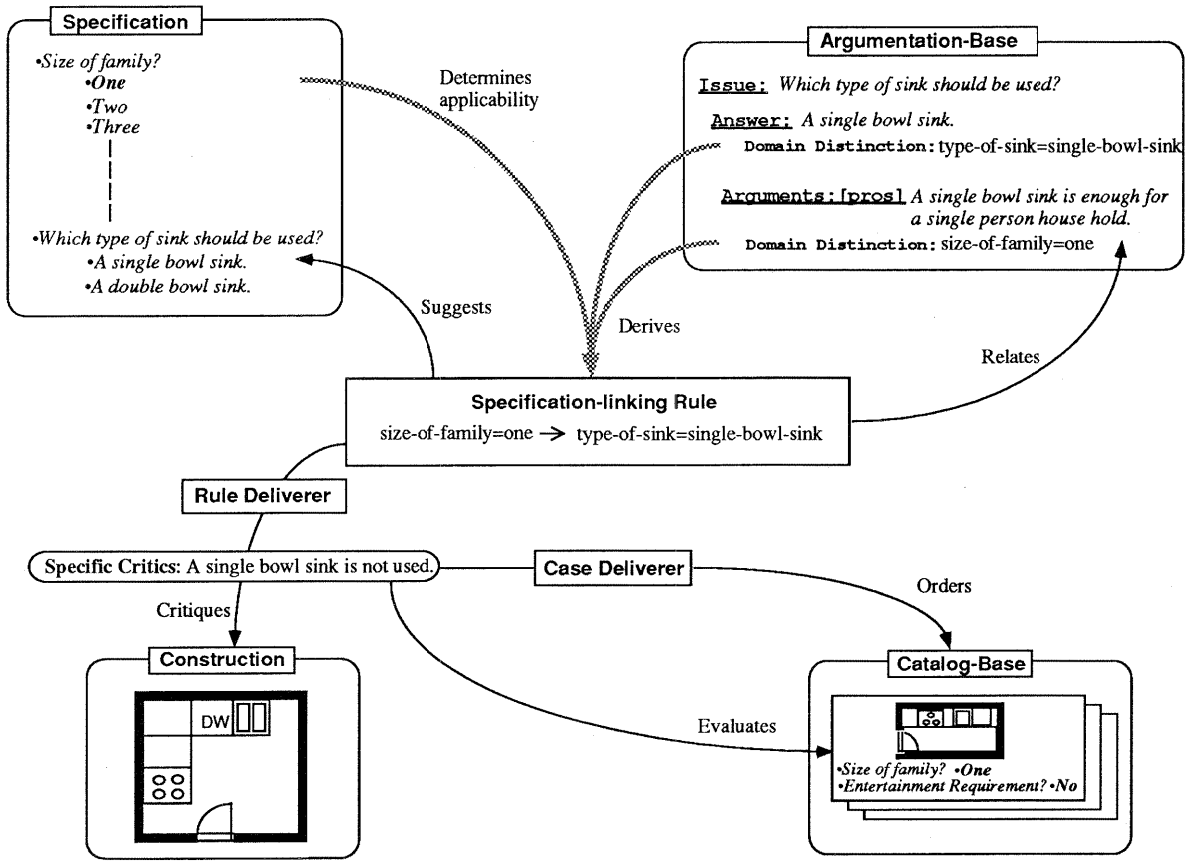


Figure 6-1: Integration of Components in KID

Specification-linking rules are derived from the argumentation, and a partial specification determines the applicability. Derived specification-linking rules are used (1) to make suggestions in KIDSPECIFICATION, (2) to show a related argument in the argumentation base, (3) to identify relevant critics as specific critics by RULE-DELIVERER, (4) to evaluate a catalog example using the specific critic, and (5) to order the catalog examples by CASE-DELIVERER.

tural specifications, the rules can be used to check compliance between a partial specification and a partial construction. RULE-DELIVERER uses the specification-linking rules to identify relevant specific critic rules.

Thus, the specification-linking rules provide links among different representations of design objects. Figure 6-1 illustrates how the specification-linking rules integrate the components of KID. Designers do not have direct access to the specification-linking rules, but consequents (the right-hand side of rules) of the rules are represented to designers in the form of suggestions in KIDSPECIFICATION, specific critic messages provided by RULE-DELIVERER, and ordered catalog examples provided by CASE-DELIVERER. Specifically:

- in KIDSPECIFICATION: to suggest how to answer other questions
- in KIDSPECIFICATION: to allow designers to play what-if games by providing consequent suggestions if one selects a certain answer
- by RULE-DELIVERER: to determine which critic rules have to be enabled based on the current specification (i.e., *specific critics*). Some specification-linking rules are related to features in construction. Such rules, then, can be used to identify critic rules that are applicable if certain answers are selected in KIDSPECIFICATION.
- by CASE-DELIVERER: to compute the appropriateness of each catalog example according to the current partial specification. CASE-DELIVERER applies a set of identified specific critics by RULE-DELIVERER to each catalog example, computes an appropriateness value of the example as the weighted sum, orders them according to this value, and presents the examples to designers.

In the following subsections, first I describe data structures of the issues, answers, and arguments in the argumentation base on which KIDSPECIFICATION is built. Next, I describe how a specification-linking rule can be automatically derived from the argumentation base. Finally, I describe how to synthesize consequents of the derived specification-linking rules in order to deal with emerging conflicts. As stressed before, KID's knowledge base does not aim at eliminating conflicts by the system itself, but rather delegates decisions to designers. Using weights assigned to each question-answer pair in KIDSPECIFICATION, the system computes the importance of each specification-linking rule.

6.3.2. Data Structure of the Underlying Argumentation Base

KIDSPECIFICATION is built on on the argumentation-base in KID. The KIDSPECIFICATION system is evolved from the REFLACT system [Fischer et al. 91b], an issue-based hypermedia system that dynamically maintains the issue-resolution dependencies. In KIDSPECIFICATION, each issue, answer, and argument is implemented as a CLOS object. There are three classes, `issue`, `answer`, and `argument`, which are all subclasses of a class `node`. The `node` class provides methods necessary for dealing with display and maintaining links between nodes.

Figure 6-2 shows definitions of the issue, answer, and argument about the selection of a type of

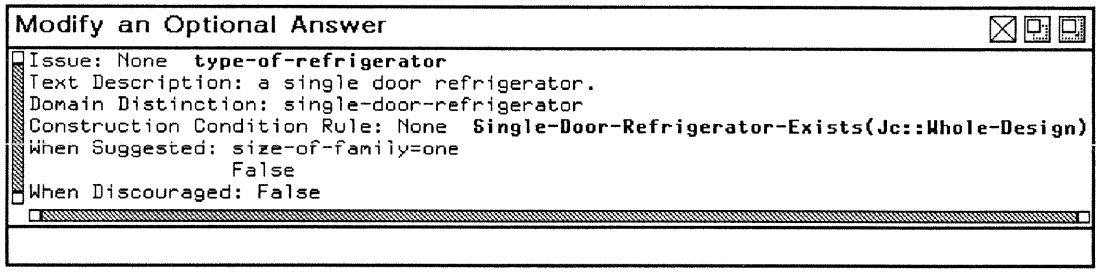


Figure 6-5: Property Sheet for Modifying an Answer
A property sheet for modifying or creating an answer.

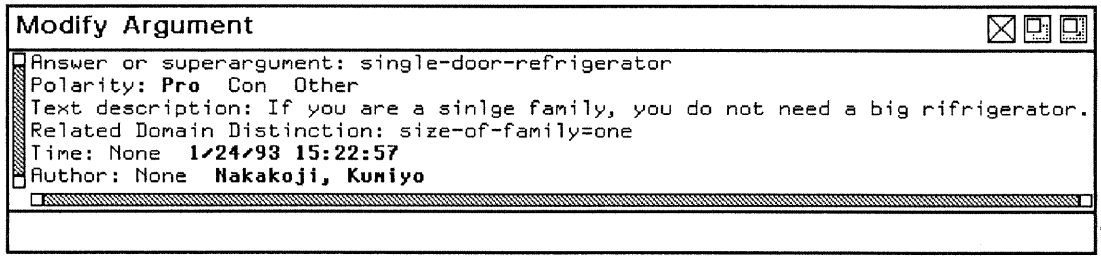


Figure 6-6: Property Sheet for Modifying an Argument
A property sheet for modifying or creating an argument.

refrigerator, and Figure 6-3 shows how the corresponding objects are presented to designers in KIDSPECIFICATION. In KIDSPECIFICATION, designers can create, modify, and delete questions, answers, and arguments. Figures 6-4, 6-5, and 6-6 present property sheets with which designers can create and modify each of these. The slot values shown in the property sheets correspond to the definition of each class object.

Each question and answer has a unique name as an identifier. For example, the question *"Which type of refrigerator do you need?"* has a name *"type-of-refrigerator,"* and the answer *"Single door refrigerator"* has a name *"single-door-refrigerator."* A pair of the identifiers of an issue and answer is called a domain-distinction, which constitutes vocabulary over the domain [Winograd, Flores 86].

The underlying formalism of KIDSPECIFICATION is a network structure of nodes, consisting of an issue, optional answers, and associated pro or con arguments. In order to form a structural network among the questions, each question must have at least one super issue. The root issue is called *"Kitchen Specification."* The *" Text-Description"* slot describes how the question is presented in the *Question* pane. The *Name* slot provides a unique identifier for the question. The *Type* slot value is either single-valued or many-valued, expressing whether the question can have one or more answers. The *Condition* slot determines the question's appearance. Certain questions can appear or disappear only when some answers are selected.

```
(defobject type-of-refrigerator issue
  :name "type-of-refrigerator"
  :superissues  '("feature-constraints")
  :subissues nil
  :text-description "Which type of refrigerator do you need?"
  :type :single-value
  :answers '("double-door-refrigerator"
             "single-door-refrigerator")
  :condition t)
(defobject single-door-refrigerator answer
  :name "single-door-refrigerator"
  :issue "type-of-refrigerator"
  :text-description "Single Door Refrigerator"
  :critic-rule "Single-Door-Refrigerator-Exists(Whole-Design)"
  :arguments '(single-door-refrigerator-arg-1))
(defobject single-door-refrigerator-arg-1 argument
  :text-description "If you are a single family, you do not need a big refrigerator."
  :answer "type-of-refrigerator"
  :polarity :pro
  :author "Nakakoji, Kumiyo"
  :Time 2938902977
  :related-domain-distinction (eq "size-of-family" "one"))
```

Figure 6-2: Issue, Answer, and Argument used in KIDSPECIFICATION

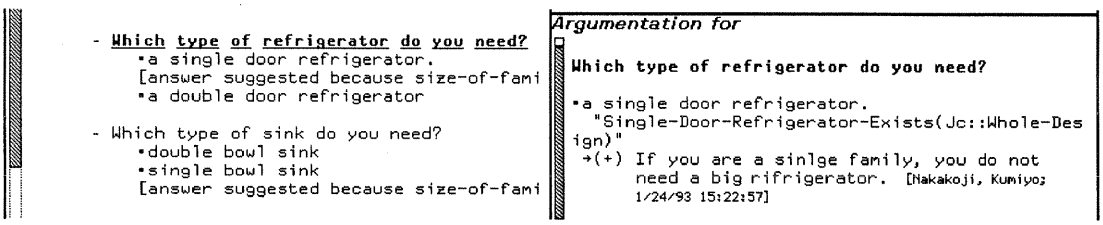


Figure 6-3: Presentation of an Issue, Answer, and Argument in KIDSPECIFICATION

A part of the KIDSPECIFICATION screen that shows how the defined objects in Figure 6-2 are presented to designers.

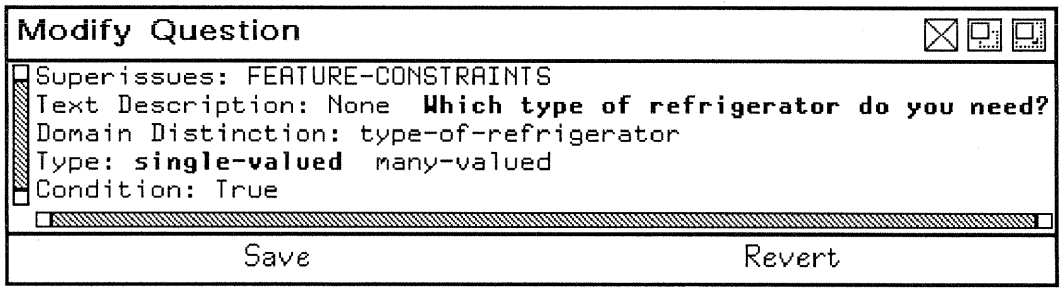


Figure 6-4: Property Sheet for Modifying an Issue

A property sheet for modifying a question. The same property sheet is used when creating a new question. The result is the definition of the issue shown in Figure 6-2.

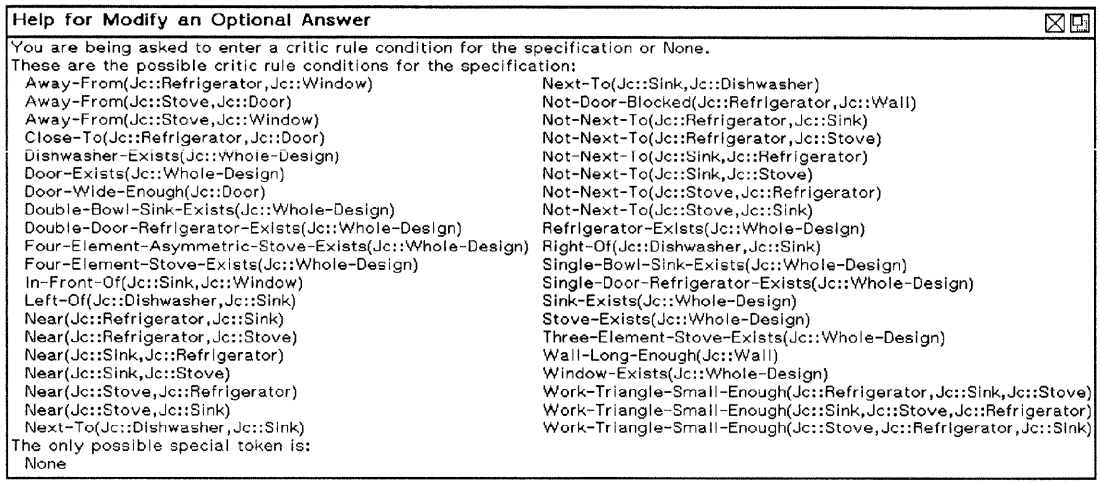


Figure 6-7: Pre-defined Critic Rules

A list of names of critic rules that are already defined in CONSTRUCTION. Designers can define a new critic using MODIFIER, which is described by Girgensohn [1992].

For defining an answer, designers have to specify to which question this answer belongs, the text description, and an identifier to be used to form a domain distinction. If the answer is related to a feature of the construction, such as a type of refrigerator, designers can associate it with one of the predefined critic rules in CONSTRUCTION. A critic rule consists of a relation and related design units, or an existence check in terms of a design unit, such as `(Away-from stove door)`, or `(Dishwasher-exist whole-design)`. The relations and the existence checks are implemented as LISP predicates over a graphical representation in the work area in CONSTRUCTION. Applying a critic rule to a partial construction replies true or false depending on whether or not the current construction satisfies the condition. Designers can define a new relation, a new design unit, and a new critic rule by combining a relation and design units using MODIFIER [Girgensohn 92].

Figure 6-7 shows all the preconstructed critic rules that KID currently has. Selecting *HELP* triggers KIDSPECIFICATION to show this list, and designers can select any rule by clicking on it. Using the associated critic rule, RULE-DELIVERER dynamically identifies *specific critics* in CONSTRUCTION.

To define an argument, designers have to specify answer or argument to which this argument belongs, whether this is confirming or negating, and the text to appear. The *Related Domain Distinctions* slot relates this argument to another issue-answer pair, and dynamically infers the dependency among the two issue-answer pairs (i.e., *Specification-Linking rules*), as described below. When they want to relate to another question, hitting the *HELP* key triggers KIDSPECIFICATION to show all the names of existing questions (Figure 6-8). Designers can pick one of them, followed by either "=" or "≠", followed by the name of one of the related answers.

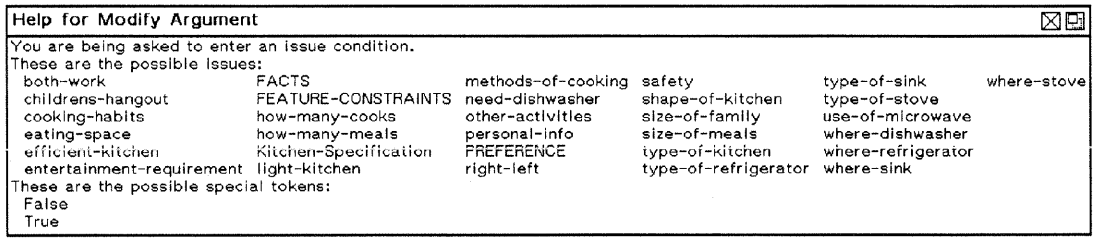


Figure 6-8: Existing Issue Names

A list of names of currently existing issues. When designers create a new issue using the property sheet shown in Figure 6-4, the defined name is automatically added to this list.

6.3.3. Derivation of a Specification-Linking Rule

Specification-linking rules are a subset of serve relationships [Fischer et al. 91b] that can be defined over the argumentation base. The serve relationships represent which issue resolution affects the resolution of other issues. For example, if you answer that the size of the household is one, then the meal size suggested is small. Specification-linking rules represent this dependency, namely, a certain issue resolution determines how to answer other issues. Some answers are related to features in the representation of constructions. In short, those rules form a decision-making network, and some of the decisions are related to surface features in the partial construction.

The approach is taken that instead of having designers define "serve relationships" in the argumentation base, the system automatically derives the dependency between two issue-answer pairs from an argument. Often, an argument to an issue-answer pair `X` is related to another issue-answer pair `Y`, and this association is made by designers filling in the *related-domain-distinction* field using the property sheet (Figure 6-6) as described above. If the argument is a pro argument, the relationship is an implication: `Y implies X`. If the argument is a con argument, the relationship is an implication of the negation of the issue-answer pair: `Y implies not X`. In Figure 6-1, for example, the argument *"A single bowl sink is enough for a single-person household"* is related to the issue-answer pair *"size-of-family=one,"* and because the argument is a pro argument for the answer *"Which type of sink do you need?: A single-bowl sink,"* the specification-linking rule *"size-of-family=one → type-of-sink=single-bowl-sink"* is derived.

The system dynamically identifies this interdependency between two issue-answer pairs and derives a specification-linking rule. Each time designers select an answer to an issue (`Y`), the system scans all the arguments to identify issue-answer pairs `X` that have arguments that are associated with the selected issue-answer pair, `Y`. Then, according to its polarity, pro or con, the system defines a PROLOG term "`require(X Y)`" or "`require(X (not Y))`." For example, from the argument shown in Figure 6-1, the following term is defined:

```
require((EQ size-of-family one) (EQ type-of-sink single-bowl-sink))
```

The system uses the PROLOG engine (as shown in Figure 6-9) to perform inference, and the maximum number of steps of inference can be determined by designers (the default is 3). This way, a newly added argument is immediately reflected in the derivation of the specification-linking rules.

```
forward-chain(_X _Y _N nil) :-
   require(_X _Y).

forward-chain(_X _Y _n0 (_Z . _ex)) :-
   _n1 is (_n0 - 1),
   _n1 >= 0,
   forward-chain(_X _Z _n1 _ex),
   forward-chain(_Z _Y 0 _void).

suppose(_X _N _then) :-
   forward-chain(_X _Y _N _ex),
   _then is (append (list '_Y) '_ex ).


backward-chain(_X _Y _N nil) :-
   require(_X _Y).

backward-chain(_X _Y _n0 (_Z . _ex)) :-
   _n1 is (_n0 -1),
   _n1 >= 0,
   backward-chain(_X _Z 0 _void),
   backward-chain(_Z _Y _n1 _ex).

why(_Y _N _because) :-
   backward-chain(_X _Y _N _ex),
   _because is (append (list '_X) '_ex).
```

Figure 6-9: PROLOG Formula for Forward and Backward Inference

PROLOG expressions used to infer specification-linking rules in both forward and backward directions. A variable _n0 is used to control the depth of inference. The result keeps the intermediate inference steps in the value _ex. The forward inference is used to make suggestions, and the backward inference is used to describe why a suggestion is made. The program is written in COMPREX-PROLOG, a pre-processor that runs on CLOS.
(COMPREX-PROLOG 2.0 Copyright (c) 4/10/92 by Alex Repenning; ABB Asea Brown Boveri Corporate Research, Hewlett Packard Switzerland, and University of Colorado at Boulder, Boulder CO. 80309.)

6.3.4. Synthesizing Consequents of Specification-linking Rules

A specification-linking rule consists of an antecedent and a consequent. Zero, one, or more specification-linking rules can be derived from a single issue-answer pair, depending on the number of arguments in the argumentation-base that are associated with the pair. When more than one inference step occurs, the number of rule consequents increases. When the system derives all the specification-linking rules from all the selected answers, the number of consequents of rules can be very large and may contain conflicts.

Designers have to deal with possibly conflicting requirements. Just providing consequents of all the specification-linking rules derived from the currently selected answers is not enough for dealing with such conflicts. A slider associated with each selected answer in KIDSPECIFICATION (Figure 5-6) allows designers to make trade-offs among the partial specification by establishing the relative importance of possibly competing or conflicting specification items [Kalay 91]. Such prioritizations should be reflected in the presentation and manipulation of specification-linking rules and their consequents.

Each consequent of the derived specification-linking rules is associated with a number that indicates the importance of the consequent. A consequent of the specification-linking rule that is derived from a selected answer with a higher assigned weight gets a higher value (i.e., more important) than one derived from answers with lower weights. A consequent of the rule that has fewer inference steps gets a higher value than one with more inference steps. Finally, the consequent that has been derived from more rules gets a higher value than one derived from a single rule.

Figure 6-10 describes the algorithm that I have built into the system for computing the value of importance of each consequent. A consequent is an issue-answer pair. When no inference is involved — that is, if the answer is selected in KIDSPECIFICATION — it gets the value 1.0. Each step of inference decrements the value by 0.1. (If ten inference steps are involved, the consequent gets the value 0.) After identifying the number of inference steps, the value is multiplied by the weight assigned to the selected answer that derived the consequent. The final value assigned to the consequent is the sum of these values. These numbers are not used as indicating absolute importance, but are used for illustrating relative importance.

Kalay [Kalay 91] described several ways to deal with synthesizing the results of design evaluations made by conflicting criteria. One way is to present the synthesized result as one value; for example, presenting the weighted sum as a result when the decision is evaluated using different criteria. Another approach is to illustrate the conflicting results and delegate the synthesis to users. For example, to use scattering plots and contingency tables as two types of representation, or to use a pie-chart visual presentation method based on dividing a circle into a number of varied degree sizes representing weighted differentiation that corresponds to the evaluation criteria. Using such pie charts has the advantage that the circle's radius can represent the normalized threshold for acceptability of the performance of the different criteria.

In KID, the approach is to assign values to each consequent, present all the consequents to designers, and have them determine how to deal with the conflicts. The values assigned to the presented consequents have no absolute meanings, but rather, should be used only as a means of comparison.

6.4. Other Approaches

In this section, I describe other approaches to support informal problem specifications. The RA (Requirement Apprenticeship) system [Rich, Waters 86] provides *"cliches,"* a library of knowledge about the particular domain of the requirement to be constructed. The system provides a set of preidentified cliches for each problem domain. Users can specify requirements using cliches, similar to filling out a template, and the system detects or infers contradictions and inefficiencies using a combination of domain-specific knowledge and predicate calculus.

Cliches are very similar to questions given in KIDSPECIFICATION. Specification is viewed as filling out a template for selecting cliches, which is the same as selecting answers in KIDSPECIFICATION. The main difference is the subsequent use of the specified information. The RA system uses the result of selecting cliches in order to automatically produce a design solution to the specification. Therefore, the main purpose of cliches are to eliminate contradictions. Our use of KIDSPECIFICATION is, in contrast, to use the information as shared knowledge between the design environment and designers so that the system can provide information relevant to the designers' task at hand. Since the design environment does not aim at producing a solution automatically, elimination of the contradictions among the specification is not the direct concern of the system.

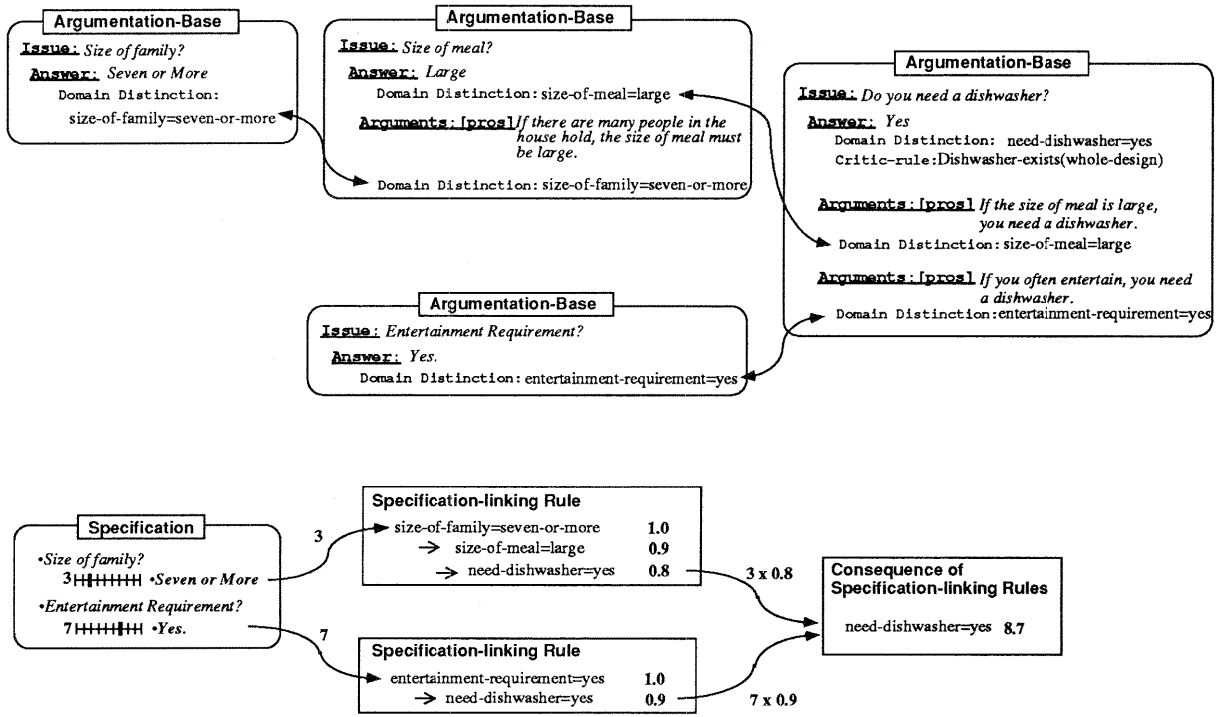


Figure 6-10: Derivation of Specification-linking Rules and Determination of Importance of Their Consequents

In this figure, the consequent *"need-dishwasher=yes"* is derived from two specification-linking rules. One is from the specification, *"size-of-family=seven-or-more,"* and the other is *"entertainment-requirement=yes."* Because the former rule includes two inference steps, the need-for-dishwasher consequent gets the value 0.8, whereas the latter rule involves only one inference, and the consequent gets the value 0.9. Each of these values (i.e., 0.8 and 0.9) is multiplied by an assigned weight to the associated selected answer (i.e., 3 to the family size, and 7 to the entertainment requirement). The sum of the results of the multiplication (i.e., $3 \times .8 + 7 \times .9 = 8.7$) is assigned to the consequent (i.e., need for dishwasher) as the value indicating importance. Note that the answer *"Do you need a dishwasher?: Yes"* is associated with the critic rule, *"Dishwasher-exists(whole-design),"* which checks the existence of a dishwasher in the current construction; thereby this critic is enabled as a specific critic and will be fired in CONSTRUCTION if the construction does not have a dishwasher.

The model discussed by McLaughlin [1991] is very similar to the specification-linking rules mechanism. McLaughlin introduced a set of qualitative associations in the form of rules between the goal predicates used to define the goal packages and the predicates used to describe the topology of a floor plan. The purpose of these associations is to allow the means by which individual goals are achieved within a specific floor plan to be identified. These rules are identified from the design recommendation text whereas the specification-linking rules are automatically derived from the recorded design rationale in the argumentation-base. McLaughlin did not provide any computational environment for managing these associations.

6.5. Summary

It is noteworthy that KIDSPECIFICATION is in no sense designed with the purpose of representing the complete problem specification. The use of symbolic representations to represent design problems is limited. Textual representations can be semantically overloaded and ambiguous. Domain-orientation of design environments reduces the ambiguity, but still it is impossible to define any textually represented concepts formally; at best concepts can be defined in terms of relationships to other concepts. No matter how many questions designers ask clients, there always exists some concepts that are missing. This is the reason why professional kitchen designers are recommended to visit clients' homes after asking the clients to fill out the questionnaires. The designers do so not only to measure walls, windows, appliances, or other dimensions of the clients' existing kitchens, but also to see how the clients have actually been using their kitchens, and to get a feeling for the clients' lifestyles; by visiting clients' kitchens, kitchen designers try to acquire the functional requirement as involvement. After designers attain these feelings, they may be able to interpret the identified functional requirements in terms of structures, goals, effects, or needs. Then the KIDSPECIFICATION system can help designers to articulate and reflect on these stated requirements.

The specification component in the multifaceted architecture provides a way to represent hidden features, such as functional or expected behavioral requirements, as well as surface features, such as structural requirements that can be represented in the construction component. Design knowledge for mapping specifications to constructions are provided by specification-linking rules. This chapter provided detail technical description of how the rules are derived. The next chapter first describes how the KID design environment works, having the specification-linking rules as a core. It then discusses how the rules are used by the two knowledge delivery mechanisms in KID.

Chapter 7

KID and its Knowledge Delivery Mechanisms

KID supports the design of architectural floor plans of kitchens from various perspectives. The system is implemented in the CLOS programming language, and runs on SYMBOLICS GENERA 8.1.

Each design session is assigned a name, which is used when storing the design into the catalog-base. In the scenario, the design session was named *"Joe-Kitchen."*

As described in the scenario, KID consists of three subsystems (see Table 4-1) — KIDSPECIFICATION, CONSTRUCTION, and CATALOGEXPLORER — which help designers' seeing-framing-seeing cycle of design processes. Reflection on a partial specification is supported with suggestions made by KIDSPECIFICATION and design knowledge delivered by RULE-DELIVERER and CASE-DELIVERER.

- KIDSPECIFICATION allows designers to specify objectives, required constraints and characteristics of the design, and to assign weights of importance to each specified item. It also provides a view into the argumentation base.
- CONSTRUCTION provides a palette of domain abstractions and supports the construction of artifacts using direct manipulation and other interaction styles.
- CATALOGEXPLORER allows designers to search the catalog space in terms of the current specification and construction.

Two knowledge delivery mechanisms are embedded in the three subsystems.

- RULE-DELIVERER identifies specific critics to be enabled to critique the current construction in terms of the current specification.
- CASE-DELIVERER orders catalog examples according to their compliance to the current specification.

Both delivery mechanisms use the information in KIDSPECIFICATION as a sense of shared knowledge indicative of the designers' task at hand. The partial specification in KIDSPECIFICATION is represented as a set of issue-answer pairs. In order to *"understand"* the meaning of these pairs and identify the information relevant to the partial specification, the system uses *specification-linking rules.*

As was shown in Figure 6-1, all of the component systems and delivery mechanisms are integrated via specification-linking rules. The specification-linking rules are derived from the argumentation base in terms of the current specification presented in KIDSPECIFICATION, thereby they are related to the arguments from which they are derived. The rules are used to suggest a selection of answers in KIDSPECIFICATION. RULE-DELIVERER uses the rules to identify which critic rules to enable as

specific critics. The identified specific critics are used to critique the current construction in CONSTRUCTION, to evaluate a catalog example in CATALOGEXPLORER in terms of the current specification, and to order catalog examples by CASE-DELIVERER.

In this chapter, I first provide a brief system description of the three subsystems; KIDSPECIFICATION, CONSTRUCTION, and CATALOGEXPLORER. Then, I describe the technical approaches of RULE-DELIVERER and CASE-DELIVERER, which use the specification-linking rules. Finally, I discuss how using the specification-linking rules for the knowledge delivery mechanisms address some issues and challenges found in research of a critiquing approach and case-based reasoning approach.

7.1. System Description

In this section, I describe mechanisms provided by the KIDSPECIFICATION, CONSTRUCTION, and CATALOGEXPLORER subsystems of KID. Descriptions of CONSTRUCTION and CATALOGEXPLORER are focused on the functionality that is related to KIDSPECIFICATION. A detailed description of the commands of KID is attached in Appendix A.

7.1.1. KIDSPECIFICATION

Figure 5-4 shows a typical screen image of KIDSPECIFICATION. The middle *Questions* window lists questions and optional answers. Designers can select one question at any time to show that their interest in the question. The *Considered Question* window keeps track of the visited questions. Designers can select any answer by clicking on it, which also automatically selects the related question. The summary of currently selected answers is presented in the *Current Specification* window. Each selected answer is accompanied by a weighting slider. Using the slider, designers can assign weights to prioritize the selected answers. The *Catalog* window lists names of catalog examples. Designers can view the specification of each of the examples. When designers select one of the names, the specification of the named example is copied to the *Current Specification* window and it can be reused and modified without deconstructing the original.

KIDSPECIFICATION offers two types of services in order to support designers in reflection on their partial specification: suggesting ways to refine the partial specification, and presenting associated issues (questions) to attend to.

1. **Making Suggestions.** The system makes suggestions regarding which answer to be selected. A suggestion is represented as an issue-answer pair, for example, *''type of refrigerator?: single-door refrigerator.''*

   - *Suggest each answer.* The system dynamically suggests some answers based on the current specification. For example, as shown in the *Question* window in Figure 5-4, the suggestion *''How many cooks usually use the kitchen at once?: one''* is made with the message *''[answer suggested because size-of-family = one]''* when a designer selects that the size of the family as one.

   - *List the Synthesized Suggestions.* The suggestions that are made based on all the selected

answers are presented as synthesized and ordered by computed values that express the importance of each suggestion (see the *Suggested* window in Figure 5-8).

- *List what-if suggestions in terms of a certain answer.* The system lists all the suggestions that would be made if designers select the answer.
- *Present the related argumentation.* Each suggestion is associated with an argument. The *Show Further Arguments* command in a menu associated with a suggestion presents the related argument to designers (see the *Argumentation for* window in Figure 5-8).
- *Describe rationale for a suggestion.* The system illustrates the inference steps that led to the particular suggestion.

The design of types of suggestions made is based on empirical evidence. Pollack [1985] observed in studies that there are three types of answers an expert provides to questioners, who often are unable to articulate what they are looking for. *Generates-type* answers are the ones that generate other answers, *enabled-type* answers are the ones that are enabled from a primary answer that was actually sought, and *alternative-type* answers alter the question itself and provide an answer inferred to be correct [Pollack 85]. Showing suggestions provides generates-type answers, and listing what-if suggestions provides enabled-type answers. Presenting the related argumentation helps designers to realize the existence of alternatives; for example, in the scenario, Jeff became aware of the existence of the question *"Entertainment Requirement?"*

2. **Locating Related Issues (Questions).** In addition to browsing the *Question* window by scrolling, designers can make KIDSPECIFICATION automatically scroll the window to present the question relevant to the task at hand. The system suggests the further refinement of the specification by locating a question from:
   - an argument provided in the *Argumentation for* window to see the related question,
   - a suggestion to see the related question of the suggestion,
   - a question stored in the *Considered Questions* window, or
   - a question presented in the *Current Specification* window to modify the current specification.

7.1.2. CONSTRUCTION

Figure 5-5 shows a typical screen image of CONSTRUCTION. Designers can construct a design by selecting a design unit in the *palette* window and placing it in the *work area* window. In the work area, designers can move, rotate, scale, and delete a design unit.

A computational critiquing mechanism in CONSTRUCTION is the main service that uses the information from KIDSPECIFICATION. Specific critics are identified by RULE-DELIVERER, one of the two knowledge delivery mechanisms provided by KID. Two types of critics exist in the system: *generic critics*, which check for violations of general design principles, and thereby are always enabled, and *specific critics*, which examine the compliance of the current construction to the current specification. Girgensohn [1992] and Fischer et al. [1993a] provide a detailed description of how critics are fired.

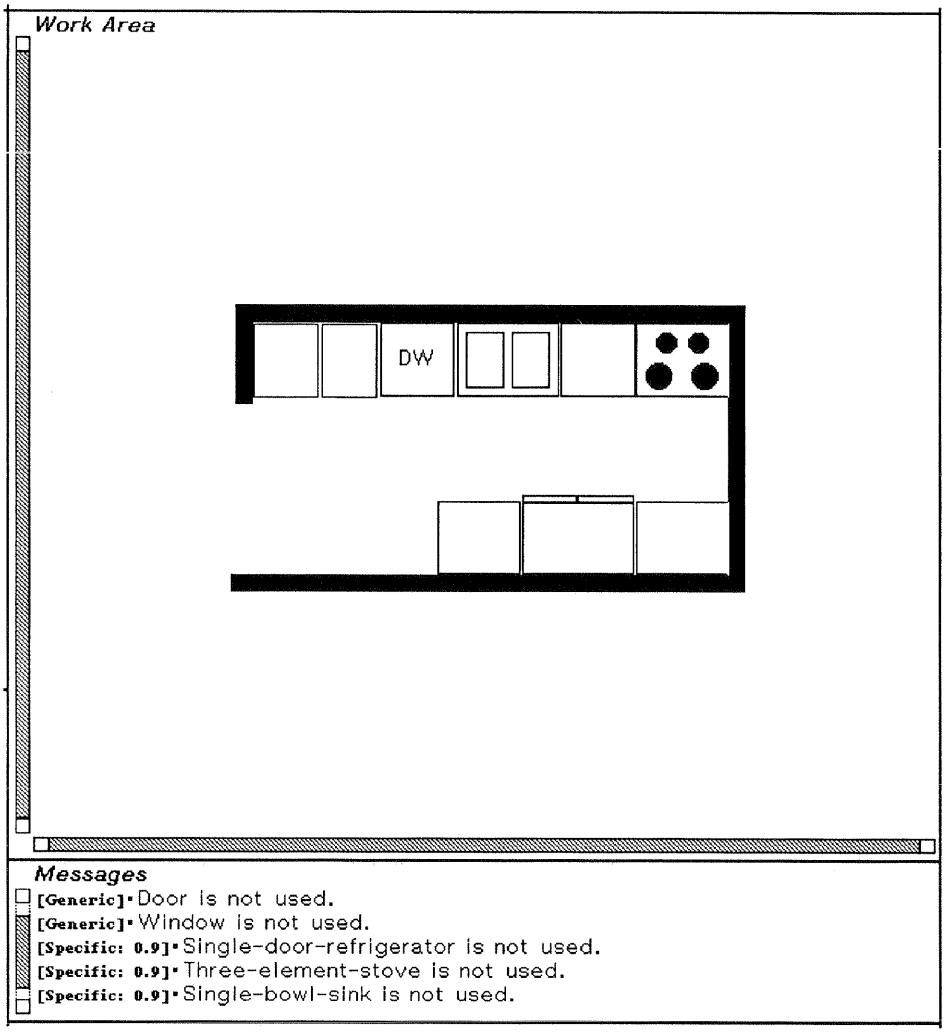


Figure 7-1: Specific and Generic Critics in CONSTRUCTION

Critics fired in CONSTRUCTION are either generic or specific. Specific critics are assigned with numbers indicating the relative importance.

The *Messages* window of Figure 5-5 presents a list of fired critics. There are two types of critics: generic and specific, as shown in Figure 7-1. A generic critic (indicated by "[Generic]") is the result of applying general design principles to the current construction, such as building codes, which are applicable to all design situations. A specific critic (indicated by "[Specific: 0.9]") is the result of analyzing the current construction in terms of the current specification. The number indicates the importance of the critic message; a higher number indicates that the critic is more important. These numbers are computed based on the weights given to the current specification in KIDSPECIFICATION.

Praises, the complement of critics, list critic rules that are satisfied in the current construction in the work area.

Designers can invoke critics in the following ways:

- by placing a design unit in the work area. The system automatically identifies violations of both generic and specific critic rules in terms of the design unit.
- by using the *Critique All* command. The system checks all the enabled critic rules, both generic and specific, with respect to the current construction, identifies violations, and presents critic messages. These critic rules include rules related to design units appearing in the current construction and rules related to the design as a whole, such as the existence of a design unit. The *Praise All* command functions the same, except it lists the satisfying conditions.
- by using the *Critique from Current Specification* command. The system fires only specific critics that are identified by RULE-DELIVERER. The *Praise from Current Specification* command functions the same, except it lists the satisfying conditions.

By clicking on the presented critic messages, designers can:

- identify design units. If a critic message is related to design units, the system blinks the related design units in the work area for a few seconds.
- locate related argumentation. The system brings designers to KIDSPECIFICATION, and presents an associated argument.
- see the rationale for the specific critique. The system explains the relationship between what the designers have specified in KIDSPECIFICATION and the critic message. The system brings designers to KIDSPECIFICATION, and presents the inference steps involved, as shown in the *Suggested* window in Figure 5-6.

7.1.3. CATALOGEXPLORER

Figure 5-7 shows a screen image of CATALOGEXPLORER [Fischer, Nakakoji 92], which supports designers in searching the catalog space. The *Matching Designs* window lists the names of the currently retrieved examples from the catalog. By clicking on a name, designers can view the example, which consists of (1) a floor plan constructed in CONSTRUCTION; (2) the specification, i.e., question-answer-weight pairs specified in KIDSPECIFICATION; and (3) the attributes, as presented in the three windows in the middle of Figure 5-7. The *Bookmarks* window holds the names of catalog examples that designers indicated were of interest with the *Add to Bookmark* command.

Design examples in the catalog are stored using the frame-based representation in the KANDOR knowledge-base [Patel-Schneider 84]. KANDOR automatically categorizes elements according to their attribute values. The attribute values of a catalog example can be edited in the interface. Appendix B shows an example of a stored case in the catalog. The CATALOGEXPLORER system is based on HELGON [Fischer, Nieper-Lemke 89], which allows designers to retrieve design examples using the frame-based search with the query-by-reformulation paradigm. A browsing mechanism for exploring the hierarchically structured catalog space is available by the embedded system TRISTAN [Nieper 85].

Figure 7-2: A Menu for Selecting an Option for Retrieval by Matching

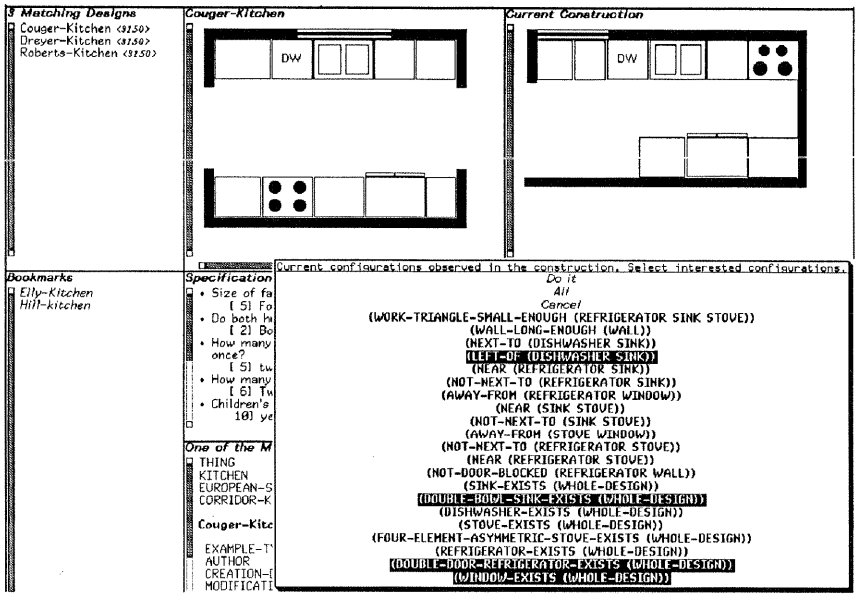


Figure 7-3: Retrieval by Matching: Construction

The system identifies characteristic features of the current construction (shown in the top right window). Designers can select any one of them and the system retrieves catalog examples that have the selected features in their constructions.

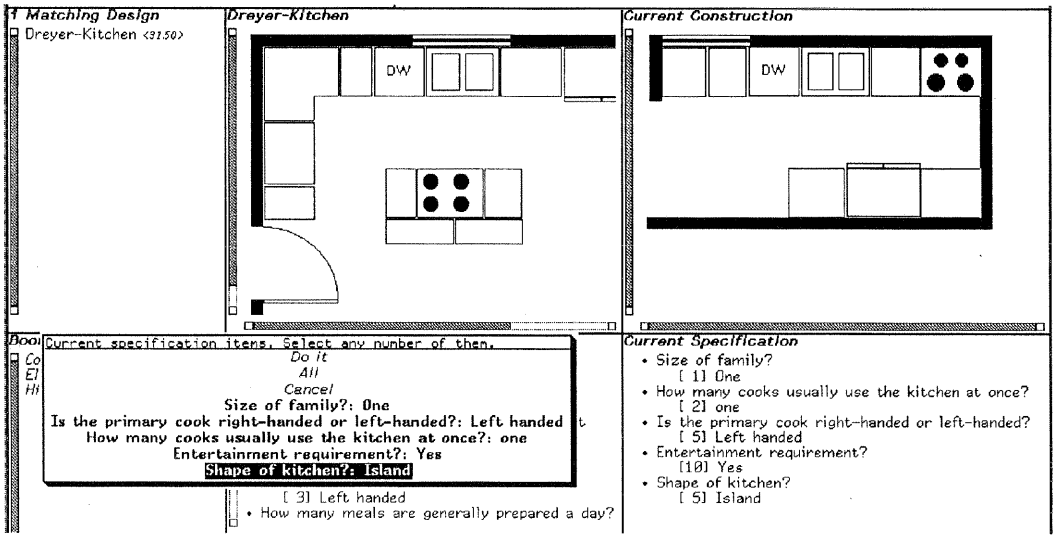


Figure 7-4: Retrieval by Matching: Specification

The system provides a menu listing the currently selected answers in KIDSPECIFICATION (as shown in the bottom left window). Designers can select any one of them and the system retrieves catalog examples, the specification of which includes the selected answers.

CATALOGEXPLORER provides three main services for designers in terms of the task at hand: (1) it invokes CASE-DELIVERER to order the catalog examples according to the current specification, (2) it reduces the number of currently retrieved catalog examples by matching, and (3) it evaluates a catalog example in terms of the current specification by applying the identified specific critics by RULE-DELIVERER.

1. **Ordering the catalog examples.**

   - The *Order by Specification* command invokes CASE-DELIVERER to order the catalog examples either from the whole catalog base, or from the currently retrieved ones using delivery rationale inferred from the current specification.
   - The *Show Delivery Rationale* command lists conditions that CASE-DELIVERER used to compute the values for ordering.

   Designers can see why a condition in the displayed delivery rationale is required by clicking on the condition. The system brings designers to KIDSPECIFICATION, and describes the inference steps to reach the condition.

2. **Reducing the number of currently retrieved catalog examples.**

   Designers can locate catalog examples that have similar features to their current design [Fischer, Nakakoji 91]. Designers can reduce the number of retrieved catalog examples in the *Matching Design* window by:

   - using the *Retrieval by Matching* command, which retrieves examples based on the match with the current construction or with the current specification. They can select one of two options (see Figure 7-2):
     - *Construction Match* (see Figure 7-3) allows designers to select some of characteristics observed in the current construction (as shown in the *Current Construction* window on the right side) presented in a menu form and to retrieve catalog examples that have the same set of selected characteristics.
     - *Specification Match* (see Figure 7-4) allows designers to select some of the currently specified items (as shown in the *Current Specification* window on the right side) presented in a menu form and to retrieve catalog examples that have the same set of specifications.

3. **Evaluating a catalog example.**

   The *Evaluate Example* command allows designers to evaluate the example in terms of the current specification. There are four options (see the pop-up menu in Figure 5-7): *Critique the Example*, *Praise the Example*, *Critique the Example from Current Specification*, and *Praise the Example from Current Specification*. They are the same functionality as *Critique All*, *Praise All*, *Critique from Current Specification*, and *Praise from Current Specification* provided in CONSTRUCTION, respectively, except that they apply the conditions over the construction of a catalog example instead of over the current construction in CONSTRUCTION. Critiqued messages appear in the window below the *Delivery Rationale* command (as shown in the middle right window in Figure 5-7). Note that the sum of the values shown in the result of *Praise the Example* is the number assigned to each catalog example, which are used by CASE-DELIVERER to order the catalog.

7.1.4. Integration of the Three Subsystems

KID is an integrated design environment consisting of KIDSPECIFICATION, CONSTRUCTION, and CATALOGEXPLORER. The integration needs to keep consistency among the information and provide smooth transitions while going back and forth among different subsystems.

The *Resume Construction* command in KIDSPECIFICATION invokes CONSTRUCTION. The *Catalog Explorer* command invokes CATALOGEXPLORER. When designers move to another system, RULE-DELIVERER identifies which critic rules should be enabled as specific critics based on the current specification. CASE-DELIVERER orders catalog examples by using the identified specific critics and computing the values of satisfying conditions for each catalog example. When invoked, both the *Catalog* window of CONSTRUCTION and the *Matching Design* window of CATALOGEXPLORER present catalog examples in the computed order.

The current specification, as shown in the *Current Specification* window in KIDSPECIFICATION (Figure 5-4), is used to derive specification-linking rules used by RULE-DELIVERER and CASE-DELIVERER, and to retrieve catalog examples to match the partial specification in CATALOGEXPLORER.

The current construction in CONSTRUCTION, as shown in the *Work Area* window in Figure 5-5, is used by CATALOGEXPLORER to retrieve catalog examples to match the partial construction.

Resume Construction in CATALOGEXPLORER brings designers to CONSTRUCTION, and *Resume Specification* brings them to KIDSPECIFICATION. In both cases, the currently retrieved catalog examples presented in the *Matching Design* window in CATALOGEXPLORER are reflected in the *Catalog* window (the same names appear in the same order). Designers can view the current specification and current construction within CATALOGEXPLORER (as shown in Figure 7-4) but cannot modify them.

7.2. Knowledge Delivery Mechanisms in KID

Pollack [1985] showed that when human experts provide advice, they do not always respond directly to the question they were asked, but try to infer a questioner's task and provide a solution that will facilitate the questioner's goal. For design environments to be cooperative with designers, the systems have to share understanding of the task with the designers.

People retrieve information in their memory based on similarity, defined in terms of goals and purposes [Minsky 91]. That is, people classify information according to what it can be used for, or which goals it can help to achieve. Having a specification component that partially embodies designers' goals, the design environment uses the shared understanding in order to retrieve "relevant" objects from the knowledge bases. We must be able to relate the structural features of each object's representation to how this object might behave in regard to achieving the designers' task at hand.

The specification component increases the shared understanding among designers and design environments. In this context, the expression *machine understanding* implies that the machine reacts or behaves according to human intuitions, expectations, or context. This is different from the use of the term *shared knowledge* by Waters [1985] in the KBEMACS project. Their "shared knowledge" is used for the system to automatically identify role distribution in performing a task between a human user and the system; then the system automatically performs parts of the task for the user. In this dissertation, shared understanding is used to identify the designers' information needs, to locate relevant information in supporting the designers' task at hand, and to present the information to the designers.

7.2.1. RULE-DELIVERER

RULE-DELIVERER identifies relevant critic rules as specific critics that are stored in CONSTRUCTION. The prestored critic rules include two types: *generic critics* and *specific critics* [Fischer et al. 92a]. Generic critic rules provide general design principles, such as building code, and are applicable to any design tasks. Specific critic rules, on the other hand, are task-dependent, and are enabled only when certain answers are selected in KIDSPECIFICATION.

As presented in Figure 6-10, some answers that appeared in consequents of the specification-linking rules are associated with critic rules stored in CONSTRUCTION (see Figure 6-7). As described above, these associations are made by designers using the property sheet shown in Figure 6-5. Figure 7-5 shows examples of definitions of critic rules. Critic rules are CLOS objects in CONSTRUCTION. A critic rule is defined in terms of a design unit. For example, the critic rule *"dishwasher-sink-rule"* is defined in terms of the dishwasher design unit. The field *":condition next-to"* is the predefined design unit relation associated with a LISP predicate over a graphical configuration in the work area in CONSTRUCTION. Girgensohn [1992] and Fischer et al. [1993a] provide a detailed description of how critics are fired.

Each critic rule has slots "generic-p" and "specific-p". If the slot "generic-p" is assigned to "t," the rule is a generic critic and is always enabled. Otherwise, the rule is a specific critic. When derived consequents of the specification-linking rules are associated with critic rules, the *"specific-p"* field of those critic rules are assigned to "t" and the rules are enabled. Whenever designers invoke CONSTRUCTION from KIDSPECIFICATION or from CATALOGEXPLORER, the system updates the slot values of all the critic rules. Specific critic rules are assigned with values indicating their relative importance, which is the same as the importance values assigned to the consequents of specification-linking rules.

7.2.2. CASE-DELIVERER

CASE-DELIVERER orders catalog examples based on the current specification. The mechanism applies specific critics identified by RULE-DELIVERER to the floor plan (construction) of each catalog example. The sum of the values of the satisfied critics rules is assigned to the example as an

```
(defobject dishwasher-sink-rule design-unit-rule
  :arguments (dishwasher sink)
  :condition next-to
  :argumentation-topic "answer (dishwasher, sink)"
  :outline-topic "issue (dishwasher)"
  :apply-to one
  :generic-p t
  :specific-p nil
  )
(defobject dishwasher-exists-rule design-unit-rule
  :condition dishwasher-exists
  :arguments (whole-design)
  :apply-to one
  :generic-p nil
  :specific-p nil
)
(defobject dishwasher-exists design-unit-relation
  :arguments (x)
  :condition (existence-check 'dishwasher)
  :directed-p t
  :critique-text "~*Dishwasher is not used."
  :praise-text "~*Dishwasher is used."
  :documentation "The design unit is used or not.")
```

Figure 7-5: Critic Rules used in CONSTRUCTION

Definitions of critic rules, a generic critic and a specific critic. The first rule is defined in terms of a dishwasher, and states that the dishwasher should be next to a sink. The *":apply-to one"* field indicates that this rule should apply to at least one sink in the current construction. If it says *"all,"* then the rule has to apply to all existing sinks used in the current construction. The second rule is defined in terms of a dishwasher, and checks whether a dishwasher exists in the current design. This rule is related to the design unit, *"whole-design,"* which is a virtual design unit created in order to check certain conditions in terms of the whole design, not just of a single design unit. The third expression is an example of the definition of the condition.

---

appropriateness value. After computing values for all the catalog examples, CASE-DELIVERER orders the examples according to these values.

Specific critics that are used to order catalog examples can be displayed with the *Show Delivery Rationale* command in CATALOGEXPLORER. For example, the *Delivery Rationale* window in Figure 5-7 lists the currently enabled specific critics with numbers indicating their importance. Which conditions of the delivery rationale are satisfied with the currently presented catalog example can be found with the *Evaluate Example* command (e.g., the *Praise* window in Figure 5-7 shows that the *Elly-Kitchen* example satisfies four conditions).

Applying a specific critic to a catalog example means checking whether the example has features of interest to designers. For example, if a catalog example satisfies the identified specific critic from the specification, as shown in Figure 6-10 (i.e., if the catalog example has a dishwasher in its construction), the example has a feature considered to be good for large-size family with a weight 3 and entertainment requirement with a weight 7.

When comparing two designs, the one that has more preferable characteristics or features is preferred to one that has fewer. In a report of user studies in intensity of preferences of various design characteristics of housing, Arias [1993b] argued for the additivity of the intensity in multi-attribute decisions. *"Intensity"* in his research was defined as a degree of preference between rank orders of alternatives, which is similar to the weights assigned to selected answers in KIDSPECIFICATION.

7.2.3. Assessment of the Two Knowledge Delivery Mechanisms

By having a shared understanding about the current task with designers, the KID design environment can make its information space relevant to the designers' task at hand. Two embedded mechanisms, RULE-DELIVERER and CASE-DELIVERER, deliver design knowledge to designers by utilizing the information given in KIDSPECIFICATION as the form of specification-linking rules.

Specific critics identified by RULE-DELIVERER provide designers with heuristic rules regarding the task at hand. These heuristic rules are linked to the argumentation base, where the designers can get further explanation about the rule. CASE-DELIVERER provides designers with previously constructed design solutions together with their problem specifications that are relevant to the task at hand. The two types of delivered design knowledge are complementary with respect to how they support designers in the seeing-framing-seeing cycle. The firing of specific critics identified by RULE-DELIVERER triggers designers to reflect on their current task. Before designers make moves that trigger breakdowns, however, the mechanism cannot support designers. Catalog examples provided by CASE-DELIVERER support designers in what to do next. Even before they start designing, looking at catalog examples gives designers some ideas where to start. Specific critics help designers at the transition from framing to seeing, and catalog examples help designers at the transition from seeing to framing.

In the critiquing paradigm, the designer's primary role is to generate and modify solutions, whereas the computer's role is to analyze those solutions and produce a critique. Different from automated design expert systems, the mechanism does not need to solve problems for designers, but to recognize and communicate deficiencies in the partial design [Fischer et al. 92b; Silverman, Mezher 92]. Therefore, the system does not have to have a complete knowledge base, which cannot exist for complex design domains.

Critiquing systems support designers in *seeing* their partial design. As Rittel [1984] pointed out, *''Buildings do not speak for themselves.''* Nonexpert designers do not have design knowledge and experience to fully understand the conversation with the materials of the situation. Even expert designers cannot have complete design knowledge because the space for relevant design knowledge is open-ended.

Critiquing mechanisms serve as ''translators'' that help designers to see and understand the ''back-talk'' of the situation. For a situation to talk back, a human has to understand the information being given. The difference between ''feedback of the system'' and ''back-talking situations'' is related to the designers' understanding. When a critic fires, reflection does not occur simply based on the message. Designers ''listen to'' the design material with the help of the translator in the form of a critic. Because specific critics are fired in terms of compliance of the current construction to the current specification, the fired critics are more related to their current task at hand, thereby helping designers to understand the feedback better.

The catalog base in KID provides a shared memory; that is, a means of maintaining the experience

of solving design tasks as design knowledge. The catalog examples are collected during the lifetime of the design environment, and the catalog base gradually evolves. The catalog base can provide a range of experience that novice designers have not had, and can even augment the memories of expert designers.

Using the catalog-base, KID can be viewed as a case-based design aiding system [Kolodner 91]. In this approach, the challenge has been how to find *useful* cases to solve a task, instead of just *similar* cases that have been studied by the analogical-reasoning research community [Kolodner 91].

There are two approaches for dealing with this case-retrieval problem. One is to assign *useful* indexes to a case at storage time to ensure that it can be retrieved at appropriate times later. In design domains, however, it is impossible to predict all the possible problem situations; therefore, it is impossible to define a set of indexes that will become useful later. The other approach is to use algorithms and heuristics to judge usefulness. CASE-DELIVERER uses specification-linking rules to retrieve (order) the cases in the catalog base. In either case, the system has to analyze the designers' reasoning goals in order to understand the designers' needs [Kolodner 91]. The specification component in KID provides a mechanism for this analysis of the reasoning goals.

The specification-linking rules are derived from the argumentation base, a collection of design rationale, which is another type of stored "case." Using the specification-linking rules as design knowledge, CASE-DELIVERER retrieves case-based information for designers that they might not have been able to access otherwise. Mapping surface features to hidden features and then using the hidden features to judge the usefulness of the design requires design knowledge and expertise [Kolodner 91; Bonnardel 91]. It is easier for designers to use surface features for retrieval, especially when they do not have enough knowledge about the task.

7.3. Summary

In this chapter, I described the KID design environment and its two knowledge delivery mechanisms, RULE-DELIVERER, which identifies critic rules to be enabled as specific critics for delivering design principles relevant to the current task, and CASE-DELIVERER, which delivers catalog examples from the catalog base relevant to the task at hand.

The specification-linking rules integrate the components and the mechanisms in KID. Both RULE-DELIVERER and CASE-DELIVERER use the specification-linking rules to retrieve design knowledge relevant to the designers' task at hand. Because the rules are derived from the argumentation base, KID can provide an explanation as to *why* knowledge delivered by the system was judged to be relevant. Designers, therefore, have access not only to delivered knowledge itself, but also to the underlying delivery rationale. Because the delivered knowledge is situated, it is easier for designers to understand the information. This way, the embedded knowledge delivery mechanisms in KID help designers learn the new design principles on demand.

In the next chapter, I present the results of user observation studies using KID. The chapter illustrates how the above claims are supported by empirical evidence.

Chapter 8

Human-System Interaction Studies

This chapter describes results and analyses of observations of experimental usage of KID. KID was tested in two ways: (1) by observing designers (i.e., users of the system) searching in a catalog base with different amounts of support provided by KID, and (2) by observing designers in designing a kitchen floor plan with KID.

The purpose of this study was not to analyze the quality of the designs produced, but to observe and analyze subjects' interactions with the system. In this chapter, I briefly describe the experimental settings and the method used in the observation. Then I present some of interesting points observed, illustrated with transcripts. I describe global observations about design, observations about the usage of KID, observed processes specifically invoked by the knowledge delivery mechanisms, and problematic characteristics of KID. Finally, I summarize these observations and draw conclusions.

8.1. Experimental Settings

Six subject groups were observed. Each subject group consisted of one or two persons, who were recruited from the community, with various degrees of expertise in kitchen design and computer usage. None of them had extensively used KID before. The subjects were given experimental tasks, test sessions were videotaped under agreement, and the protocols were partially transcribed.

8.1.1. Observation Methods

The *constructive interaction method* [Miyake 86] was used to analyze the interaction between the subjects and the system. The method uses two subjects in each group and observes them talking to each other. The method circumvents the issue of think-aloud protocol analysis [Ericsson, Simon 84], in which the subjects are forced to think aloud in a situation that would normally be silent. For testing the benefit of knowledge delivery mechanisms, observations such as whether subjects' attentions are invoked by the delivery, whether subjects are annoyed by the delivery, and what factors trigger their reframing problem specifications and solution constructions are crucial. It is difficult for subjects to report when a small shift of attention occurs. Such shifts of attention are very subtle. Even a subject may not notice such shifts, and moreover, talking aloud causes another shift of attention. In contrast, if two subjects work together, those shifts of attention are more naturally captured in talking, which is a very natural process for people engaged in a conversation.

Surveys were filled out by the subjects after the test sessions in order to capture their background in terms of the kitchen design domain and CAD (Computer-Aided Design) tools and to ask them their feelings about KID. No quantitative analyses were made in accordance with a classical experimental setting.

Table 8-1: Subject Groups in User Observation

Six subject groups were observed. *"Low-None"* means one participant had a little expertise and the other had no expertise. The *"Types of Task"* column shows the type of experiments performed — the search task or the design task described in Section 8.1.2. *"Given"* means that a design requirement was given (see Figure 8-1), whereas *"Not Given"* means that a design requirement was articulated from the subjects.

Name	Number of Participants	Expertise in Kitchen Design	Expertise in CAD tools	Types of Task	Requirement Description
A	2	None-None	None-None	Design	Given
B	2	Low-None	None-None	Design	Not Given
C	1	None	None	Search	Given
D	1	High	Low	Search	Given
E	1	None	Low	Search	Given
F	2	None-None	Low-None	Search	Given

Table 8-1 summarizes the subjects groups observed. In those groups indicating only one participant, the experimenter acted as a partner for constructive interaction to facilitate verbal interpretation of problems and assumptions of the subject. The two subjects sat down in front of the KID system running on a SYMBOLICS Lisp machine. One operated the mouse cursor and the keyboard. Because the purpose of this experiment was not to test how designers learn how to use the system but rather how designers were supported in performing a design task, the appropriate help for system usage was verbally provided on demand by the experimenter.

8.1.2. Two Types of Tasks

The experiment used two types of tasks. One was to find the "best" floor plan example from a catalog-base of KID, and the other was to design a kitchen floor plan using KID. An informal requirement description for the task was either given (see Figure 8-1), or was articulated by the subjects themselves after being told to design their own kitchen. These different conditions were set up in order to expose different aspects of problems expected to be observed during the experiments.

For the given task, the experimenter acted as a client. Figure 8-1 shows a requirement description initially given as a design task. Because one of the goals of the experiment was to observe how KIDSPECIFICATION triggers and helps subjects to uncover and understand the problem, some of the requirement conditions were initially hidden. This way, the experimenter could observe how KIDSPECIFICATION triggered subjects to reexamine their initial problem specification. The predetermined conditions were told to the subjects on request. Any questions that were not in the predetermined conditions were answered as *"I do not know."*

1. **Search Task**

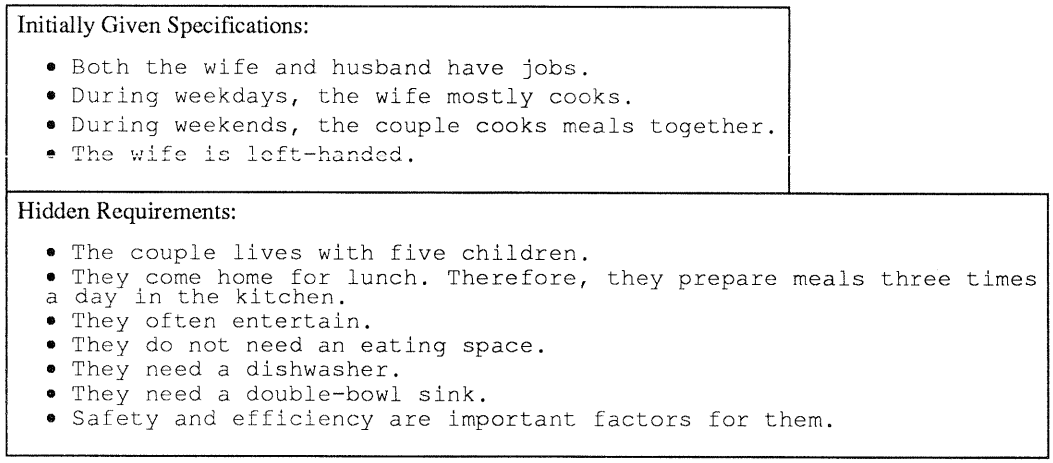


Figure 8-1: An Informal Requirement Description of the Task Given to Subjects

In order to observe how subjects gain the understanding of the design problem, some of the conditions were initially hidden. They were told to the subjects as requested. Anything asked that was not stated in the conditions were answered as *"I do not know."*

Subjects were asked to find the "best" catalog example in a catalog-base of KID in terms of the given design requirements. The catalog had thirty floor plan examples, which were initially ordered at random. They were asked to search for an example under four different conditions with regard to types of tool support they were allowed to use. At each step, tool support was incrementally provided. The different types of tool support were provided in the following order:

a. Using a scroll bar of the *Catalog* window in CONSTRUCTION.

b. CATALOGEXPLORER was described to the subject group. They were encouraged to use the *Retrieval by Matching - Construction* mechanism (see Section 7.1.3) in order to retrieve catalog examples that have similar floor plans. First, subjects were asked to construct a partial kitchen floor plan in CONSTRUCTION. Then, subjects could retrieve matching examples in CATALOGEXPLORER. While using CONSTRUCTION, critics sometimes fired to critique the current construction.

c. KIDSPECIFICATION was introduced. Then, the *Retrieval by Matching - Specification* mechanism (see Section 7.1.3), which retrieves catalog examples that have similar specification (selected answers) was described. The mechanisms of making suggestions were sometimes used and specific critics were identified by RULE-DELIVERER in order to provide information to the subjects.

d. Finally, subjects were presented with catalog examples automatically ordered by CASE-DELIVERER in accordance with the partial specification. The *Evaluate Example* command, which evaluates a catalog example in terms of the current specification using *Specific Critics* was introduced.

Subjects were given instructions about how to use each mechanism immediately prior to performing the search task. For each condition, the same requirement description and the same set of catalog examples were used. Subjects were allowed to ask questions regarding the given design requirement as a design task.

2. **Design Task**

   Subjects were asked to design a kitchen floor plan using any of the existing mechanisms of KID. Prior to the experiment, a short training session was provided in order to describe the mechanisms. Requirements for the design task were either given by an experimenter or gradually articulated by the subjects themselves after being told to design their own kitchen.

KID has a large functionality with quite complex interfaces. The system is not intended to be a *walk-up-and-use* system and subjects needed to spend 20 to 30 minutes in understanding the functionality of the system. The search task that was performed iteratively with a gradual introduction to the systems' functionality provided a nice training session for the subjects.

Interestingly, many subject groups who started with the search task started to construct kitchen floor plans at some point during the search, and in the end, they had unconsciously switched to the design task. Many started to construct a floor plan when they were asked to use the retrieval-by-matching:construction mechanism because it required them to construct a floor plan in order to retrieve catalog examples. Consequently, in both tasks, subjects interacted with and reacted to the system in the same manner. Therefore, I present the combined results observed in the both tasks.

8.2. Observations About Design

In this section, I present observations about design, which are not necessarily related to the specific mechanisms provided by KID. They include:

- survey results,
- problems of SYMBOLICS user interface styles,
- multiple interpretations of the problem,
- psychological dependency on the system, and
- design style as cultural differences.

8.2.1. Survey Results

Figure 8-2 summarizes the result of the surveys filled out by the subjects after the test sessions. It is noteworthy that all but one of the subjects (the expert) answered that they thought they designed a ''better'' kitchen using KID. This indicates that KID provided subjects with design knowledge about the kitchen design domain to which they otherwise did not have access. None of the subjects felt the system's behavior to be intrusive. Some of the subjects felt the system was too complex. This complexity may have made them answer that they would not use KID for designing their own kitchen in the future (see the last question in Figure 8-2).

Have you designed a kitchen before? Yes (xx) No (xxxx)

If "yes", then how do you categorize your expertise in the kitchen design domain? Expert — Novice

Have you used any CAD (Computer-Aided Design) systems before? Yes (xxx) No (xxx)

If "yes", then how useful did you find the KID system compared to the tools that you used before? More Useful — Less Useful

How useful did you find the KID system? Very Useful — Not Useful

How useful did you find the each mechanism in searching for a catalog example?

1: Browsing window in the CONSTRUCTION component: Very Useful — Not Useful

2: Retrieval by Construction Match in CATALOG-EXPLORER: Very Useful — Not Useful

3: Retrieval by Specification Match in CATALOG-EXPLORER: Very Useful — Not Useful

4: Order by Specification command in CATALOG-EXPLORER: Very Useful — Not Useful

5: Critic fired in CONSTRUCTION: Very Useful — Not Useful

Which mechanism of the above five do you like the best? 1 2 3 4 5

Do you think you would have designed a kitchen floor plan differently if you had not been using KID? Yes (xxxxx) No (x)

If "yes", then how good would the design have been compared to the design that you created in KID? Worse — Better

How did you like the system's automated behavior (such as critics)? Like — Dislike

Did you think the KID system was collaborative with you while performing a design task? Very Collaborative — Not Collaborative

How do you characterize the KID system?
(select any number of them)

Intrusive	Beneficial (x)	Friendly	Useful (xxx)	Collaborative (x)
Insightful (xx)	Intelligent (x)	Complicated (xxx)		

Are you interested in using KID if you need to design your own kitchen in the near future? Yes — No

Figure 8-2: Summary of the Result of the Surveys

Six surveys were collected. "x" represents one count. Some subjects did not answer every question.

8.2.2. Problems of SYMBOLICS User Interface Styles

The SYMBOLICS LISP machine on which KID is implemented has unique characteristics in its user interface, such as a mouse documentation line located at the bottom of the screen that always displays information about which mouse button is assigned to which functionality. Because none of the subjects were SYMBOLICS experts, the following problems were found:

- In using a scroll bar: scrolling up and down is done in a different manner than more common styles such as the Mac user interface.
- In the tiling window system: the windows are tiled on the SYMBOLICS display, and no overlapping is allowed except for property sheets and a pop-up menu. Some subjects wished they had been able to kept a part of the previous screen somewhere on the display to remind them of what was there.
- Slow response: this was a very serious problem. While the system was displaying information as delivery, sometimes the subjects had to wait for from thirty seconds to a couple of minutes. This frustrated some of the subjects. Even if the system delivered information at the *right* time in terms of the context, if subjects had to wait for more than a minute, the timing seemed to be not right any more.

8.2.3. Multiple Interpretations of the Problem

Given the same informal requirement description (see Figure 8-1), different subject groups answered the same question (e.g., *''How many cooks usually use the kitchen at once''*) differently (e.g., *''one''* or *''two''*). The following two transcripts from different subject groups illustrate this point.

> **[1]:** [reading a question] *How many cooks usually use the kitchen at once? .. Ummm... "usually" just one but weekends its two.... ummm... but it would not hurt to say two. Isn't it?*
>
> **[2]:** [after reading the question] *usually.. I don't know. it says.. does not say occasionally.. so maybe I have to say one.*

Both sound reasonable, but one selected an answer *one*, and the other selected *two*. Because kitchens for one cook and those for two cooks require different sets of conditions, the decision affected the systems' subsequent suggestions.

The wording *''usually''* in the question might have been a main cause of this ambiguity, but the problem is, to some extent, inevitable. As discussed in Chapter 2, the meaning of text cannot be completely defined without taking into account its usage context. The above phenomena clearly illustrate the problem.

8.2.4. Psychological Dependency on the System

Although KID was designed to be a cooperative assistant for designers, some of the subjects in the study had started to depend on the system. For example, this phenomenon was found in the following transcript.

> **[3]:** [In KIDSPECIFICATION, while selecting answers] *I do not see any suggestion for the type of refrigerator, so I leave it unmarked.*

This group had started being immersed in how to satisfy the system (i.e., not to fire any critics) instead of thinking about the functions of the kitchen.

The same type of observations were found in the search task. Some subject groups tried to make KID suggest the one that the subjects had already selected as the best by browsing the catalog. When the system suggested a catalog example that looked totally different from the one they had previously selected, the subjects tended to become frustrated.

8.2.5. Design Style — Cultural Difference

One group of subjects, who were from Japan, viewed designs from slightly different perspectives than did the American subjects. For example, they did not like to have a stove in the island, because

> **[4]:** *when you deep-fry or something, it is really dangerous to have a pot full of hot oil in the middle of the kitchen, especially if you have small children around.*

In Japan, deep-frying is a common method for cooking. As discussed below, mentally simulating a usage situation was the common method to evaluate a floor plan. Such usage situations differ based on cooking styles, such as what they cook and how they cook. For example, in Japanese cuisine, cooks generally spend much a longer time cutting food before heating than in the Western style cuisine. Therefore, having a space for a cutting board between the sink (where they prepare food) and stove is very important. Consequently, the Japanese subject group selected a different kitchen example as the best example, which none of the other subjects groups had chosen.

This indicates that the system should allow designers to use a different set of design knowledge such as critics, design units, or catalog examples, in order to represent different perspectives [Fischer et al. 92a].

8.3. Observations about the Usage of KID

In this section, I describe observations made in terms of the usage of KID, including:

- seeing and mental simulations,
- mapping between hidden features and surface features,
- asymmetry in coevolution of specification and construction,
- different importance of arguments,
- dealing with a weighting scheme,
- the role of the *Critique All* command,
- the role of positive feedback, and
- use of catalog examples.

8.3.1. Seeing and Mental Simulation

Mentally simulating kitchen usage was the main method that all the subjects used for evaluating the kitchen floor plan with regard to abstract functional requirements (see Section 6.1).

> **[5]:** [while examining a catalog example] *This location of sink, stove, refrigerator is good because if somebody walks into the kitchen and talks to you, they still do not interfere with you... and it allows somebody to help you if you want to.*

> **[6]:** [looking at another example] *The refrigerator right next to a sink... ummm. .. If you're in front of a sink working, your elbows .. keep bumping.. also, you have a potential hazard here if a pot handle sticks out. So this one I would not use it at all.*

The following transcript presents a typically observed seeing-framing-seeing cycle (see Section 2.4). Again, mental simulation was used in order to map abstract functional requirements to structural constraints in a floor plan, and vice versa. By iterating these processes, the subjects gained an understanding of the problem.

> **[7]:** [looking at an example] *So first of all, you have a two-person cooking in the kitchen situation, plus you have a guest in the kitchen while you are cooking, and also you have... a... one-person cooking situation. So here you have a space in between here for two people to share the cook top while you are doing with a sink or a refrigerator. Ummm .. you could also, ... you can have a microwave anywhere along here.*
> [asked by the other partner] *so you assumed that they need a microwave.*
> *Yeah... when you come home from work, do you cook, or cook on weekends. If you cook on weekends then you need just re-heat what you have.. and normally... . how would you do that?* [the partner said microwave] *So you need to find a spot for a microwave so that somebody can use that. And be out of traffic pattern of other person.*

In this transcript, first the subject tried to organize the given requirements (see Figure 8-1) in her head, and then mentally simulated the usage situation in terms of the example floor plan. While doing it, she discovered that they might need a microwave oven, which was not a part of the requirement specification before. Thus, the subject reframed the problem as gaining the understanding the design problem.

8.3.2. Mapping Between Hidden Features and Surface Features

While the subjects were selecting answers in KIDSPECIFICATION in terms of *surface features* such as selecting a type of refrigerator, they constantly went back and forth between the identified hidden feature requirements (i.e., the size of family) and given alternatives in surface features, using the mental simulation mentioned above.

> **[8]:** [the subject was looking at alternative answers given in KIDSPECIFICATION] *Type of refrigerator... First of all, what type of food do they buy from the store at a time ? lot of fresh fruit, vegetables? .. so they buy large quantities. ....And I would assume that a double door refrigerator has one for a freezer, and one for a refrigerator, so if you buy a lot of frozen products, then we probably need a good size of freezer...*

> **[9]:** [looking at another question] *Type of sink.. two cooks there, and they may be mixing a salad or rinsing the vegetable, and if the other cook wants to wash something, then she needs to move all the vegetable.. Umm.. so they preferably need a double bowl sink.*

As presented in this transcript, the subjects gradually gained an understanding of the problem by identifying functional requirements, and developed the solution construction (structural requirements) at the same time.

8.3.3. Asymmetry in Coevolution of Specification and Construction

The subjects coevolved the problem specification and solution construction. An initial assumption was that the subjects develop specification and construction concurrently starting from a small part of each of them. In both the search task and the design task, however, an asymmetry was found in the development of specification and that of construction. The subjects first filled in almost all the answers in KIDSPECIFICATION, then started to search the catalog or to construct a floor plan. On the other hand, in CONSTRUCTION, they gradually developed a floor plan while often interrupting themselves by attending to related design information such as critics or catalog examples.

In summary, subjects could frame quite a large part of the specification at the beginning, compared to a construction. Then, they gradually modified the partial specification. The degree of completeness of an initial specification is much larger than that of the construction, but the speed, or the amount of changes made, of the development of specification is smaller than that of construction. This result is probably due to the simple design task that was used in the experimental setting, and the complexity of the system, especially for the subjects who had not used KID extensively before.

8.3.4. Different Importance of Arguments

Some of the arguments were found more critical than others, which affected the decision making.

> **[10]:** [the displayed argumentation said *"if you are a left-handed, a dishwasher should be on the left side of a sink"*] *I would argue against that. Because you are left-handed, and when you picked up the dish in the sink, and wipe it off with left-hand while holding it with your right-hand, and if the dishwasher is on the left side, then you are reaching over like this. So, then, if you are left-handed, and if the dishwasher is on the right side, and then it makes it much simpler.*
> [pause for a second]
> *ummm.. but ... normally you are limited to a space, and you do not have a choice to .. so it is ridiculous to argue about this. ... ummm.. Normally people are so getting used to the location of the dishwasher on the one side over the other, then it just doesn't matter to them. You're not going to change their mind. And that's fine. There is no harm whether what the rule says.*
> *ok, there is several ..... umm.. there are a couple of considerations here. One thing, you don't want a dishwasher door coming down and blocking access to .. say, a cabinet where you store most of your dishes, so that you have to take the things out of the dishwasher, and put them on the counter, and then to the cabinet because with the door down you couldn't get to the cabinet. So, there is consideration like that ... that would come before right or left hand placement.*

As presented in this transcript uttered by the expert, first she articulated an argument about location of the dishwasher in terms of a sink. While she was describing the argument, however, she remembered a *more critical* consideration — the location of the dishwasher in terms of the cabinet where most of dishes are stored.

The transcript indicates that some dependencies are more important than other dependencies. And this differentiation may be accompanied by another argument, such as *"People can get used to whether the dishwasher is on one side of a sink or the other."*

Although all the specification-linking rules have the same initial weight unless more than one step of inference is involved, KID needs to provide a mechanism that allows designers to specify the relative importance of each argument, which will then be reflected in determining the weights of specification-linking rules themselves.

8.3.5. Difficulty in Dealing with a Weighting Scheme

The transcript below presents an example of how the subjects assigned a weight to a certain specification item.

> **[11]:** [a subject was looking at the question *"do you need a microwave oven?"*] *Oh, we did not get to this question... Do they want a microwave oven?*
> [the experimenter answered *"I don't know."*] *Then,.. Umm.. they are in a hassle when they come back from work, aren't they? Then it's nice to have a microwave. But since they do not know it for sure, let's put the weight.. let's say... 2.*

On the other hand, some subjects had difficulty in assigning weights, especially when they were asked to compare abstract ideas such as the number of cooks who uses a kitchen and a need for an eating space. Once the subjects got the contradictory suggestions in the context, then they could answer, or assign weights. But until then, the subjects seldom modified a given default weight.

This indicates that providing the context (i.e., information such as getting this implies getting that) supports designers in making trade-offs. Making seemingly independent decisions is difficult for designers. The specification-linking rules, which are provided in the form of suggestions in KID, make such hidden dependencies explicit, and thus help designers in selecting answers and deciding the relative importance of the selected answers.

8.3.6. The Role of the *Critique All* command

"Am I done?" or *"How can I improve my design?"* was often uttered by the subjects. They used the *Critique All* (or *Critique from Specification*) command to see what the system would say. Often, a few critic messages were presented to the subjects. Then, they carefully examined these messages one by one, and when they did not agree with any of the critics, they were satisfied and felt that they had finished the task. When the subjects found one of the critique messages reasonable, they kept working on their design by trying to address the problem identified.

The *Critique All* command, or other passive types of critiquing mechanisms that are invoked explicitly by the subjects also helped the subjects by indicating how they could improve their current design. As discussed in Section 8.5, some subjects complained that there was no support in identifying how they could improve their design in terms of delivered catalog examples.

As Rittel and Webber [1984] pointed out, design as a wicked problem has no stopping rules or criteria that tell when *"the"* or *"a"* solution has been found. During the study, this was very typical with all the novice subjects. They often needed someone else's opinion as to whether they were done. Computational critic mechanisms were influential in convincing designers that their design task was completed.

8.3.7. The Role of Positive Feedback

None of the subjects used the *Praise All* or *Praise from Specification* command. They did not request positive feedback from KID.

On the other hand, when provided, a serendipitous positive feedback helped the subjects to trigger the reflection.

> **[12]:** [they were about to complete the design task] *Do we have anything else to put in there? Are we finished?*
> [the other partner said] *I don't know....*
> [while selecting the *Critique all* command] *Let's find out what the system will say.*
> [when the system displayed *" kitchen satisfies all the design principle,"* the other partner mumbled] *but I do not understand... the door of the refrigerator is interfering with the door.*

In the above transcript, first the subjects were not sure whether they were done with the design task, but they still did not know whether the design still had a problem. When the system *praised* the design with the complimentary statement, however, one of the subjects immediately become aware of a problematic situation in their design. The system did not point out the problem, but only triggered the subjects to reflect on the design.

8.3.8. Use of Catalog Examples

In the study, subjects always examined delivered catalog examples carefully before deciding whether the examples were useful for their tasks. One of problems in using cases — that people tend to assume that an answer from a previous case is right without justifying it in the new problem [Kolodner 91] — was not observed at all. The subjects often did not like the system's delivered catalog examples and wanted to justify and evaluate them. The possible explanations for this include: (1) the current specification and the catalog's specification had never been matched perfectly, and (2) the subjects know that the relationships between the problem and the solution were very subjective and arguable.

8.4. Analyses of Knowledge Delivery Effects

In this section, I describe phenomena observed after the subjects looked at the delivered knowledge by KID. *Specific Critics* provided by RULE-DELIVERER present critiquing messages for the current construction, or as a result of evaluating a catalog example. CASE-DELIVERER presents ordered catalog examples according to the current specification. The *Show Delivery Rationale* command lists critic conditions used to order the catalog. In the study, it was found that when KID delivered such design knowledge, the subjects ultimately responded either to the associated arguments, or to the catalog example.

When presented with the critics, the initial reaction of the subjects was to ask for further explanations as to why the critics were significant, except in a few cases such as when the subjects were already quite familiar with the critic message (i.e., many of the subjects knew what the *work triangle* means). Instead of arguing for or against the fired critic message itself, the subjects agreed or disagreed with the associated argument. For example, when the critic *"the refrigerator should be near a door"* fired, the subjects neither said, *"yeah, the refrigerator should be"* nor *"no, it should not be,"* but they looked at an argument for this critic (i.e., *"so that incoming cold food can quickly be stored in the refrigerator"*) and said,

> **[13]:** *No, because a garage is not next to a kitchen, right? You have to carry groceries anyway.*

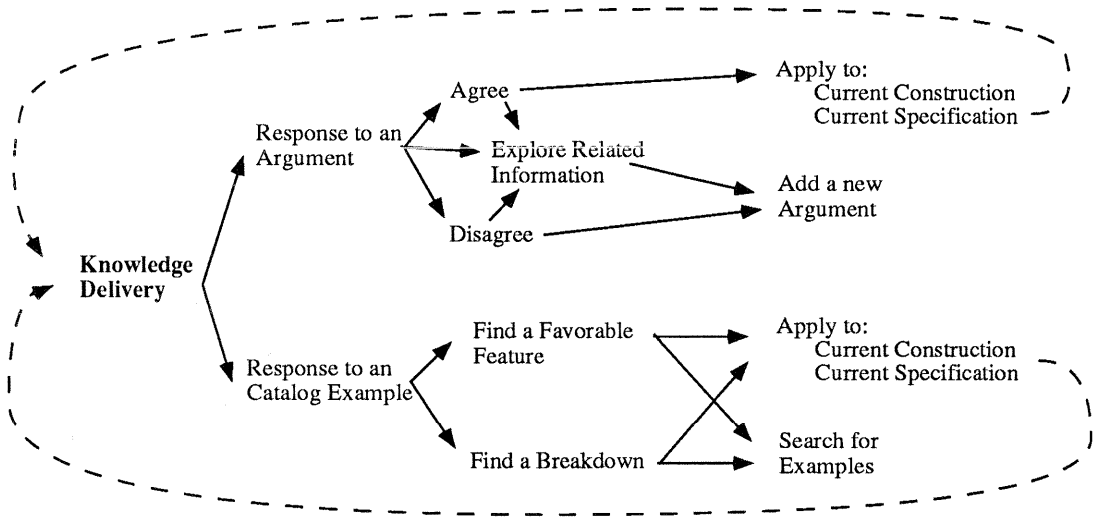


Figure 8-3: Classification of the Effects of Delivery

The subjects' reactions to the system's delivery are classified. When argumentation was delivered, the subjects either agreed or disagreed, and different reactions were made accordingly. When catalog examples were delivered, the subjects either liked or disliked one of the examples, depending on whether they discovered a good or unknown feature or an unfavorable feature in the example. Application of the delivered knowledge to the current design resulted in the next knowledge delivery, and thus formed the cycle of processes.

When presented with the ordered catalog examples, the subjects either started to examine the construction and specification of the example at the top of the list, or they asked for further explanations about why KID judged the example as the best for their specification using the *Show Delivery Rationale* command, and then accessed to the underlying argument associated with the listed critic conditions. There were few cases when the subjects did not pay attention to the ordered catalog; that is, when the catalog example ranked at the top of the list as the most appropriate was the same (or a very similar) catalog example that the subject group had already chosen with previous conditions in the search task.

When presented with the delivered information, the subjects did not seem to feel as if they were searching the information spaces. From the subjects' points of view, what KID delivered was two types of design knowledge: argumentation and catalog examples. Critic messages or a list of delivery rationale were just intermediate representations that brought them to the design knowledge in the argumentation-base.

Figure 8-3 summarize types of responses, or effects, made by the subjects when KID delivered knowledge. The classification in the figure is very simplified. The reactions were intermingled and it was sometimes hard to identify clearly the cause-effect relationships.

In this section, I first present two transcripts illustrating typical intermingled responses. Then, I present short transcripts that illustrate the classification represented in Figure 8-3.

8.4.1. Two Case Studies

The two transcripts below present typical flows of design processes after KID delivered the knowledge to the subjects. The first one illustrates how the subjects' differing reactions depend on their agreement to the given argumentation. The second transcript shows how delivered catalog examples triggered the subjects to reflect on their partial construction.

Case 1. The subject was performing the search task, and he already had chosen *Isak-Kitchen* as the best example to the given task by browsing. Although it was a search task, he had already started to construct a floor plan in CONSTRUCTION by copying *Isak-Kitchen* to the work area so that he could modify it. In CATALOGEXPLORER, CASE-DELIVERER suggested another example. The subject wanted to know why his selected example, *Isak-Kitchen*, was not suggested.

> **[14]:** *Can I see what's wrong with this [Isak] kitchen?*
> [He used the *Critic From Current Specification* command. The system presented a critic *"a double-door refrigerator is not used."*]
> *Why so ...?*
> [He asked for the explanation, and found the argumentation *"if you often use a microwave, you probably need to store a lot of frozen food, thereby you need a big refrigerator."*]
> *Oh, because I answered I need a microwave* [in KIDSPECIFICATION]. *OK, then probably you need to change that* [in CONSTRUCTION]. *That's a good point. Actually.. the other thing was..* [he was reading another critic message] *The three element stove was not used ... Why three element?*
> [He asked for the explanation, and found the argumentation *"if you always use a microwave oven, you do not need many elements of a stove."*]
> *Nah,.. this isn't the point. We do not know whether they use a microwave for sure, so the system should not complain. ...*

With the delivered information (ordered catalog examples), the subject was motivated to explore more information because the system's suggestion (another catalog example) and his judgment (*Isak-Kitchen*) were different. He evaluated his favored kitchen and accessed to the domain knowledge (the two arguments).

The subject reacted differently to the two arguments based on the same specification item (i.e., use of microwave). First, he liked the argument underlying the double-door refrigerator critic. Therefore, he was convinced and decided to modify his current construction, which had a single-door refrigerator. On the other hand, he did not agree with the argument behind the three-asymmetric-element stove critic. The argument was based on his microwave specification, and he attributed his disagreement to the fact that *"we do not know whether they use a microwave for sure,"* whereas he liked the previous argument based on exactly the same microwave specification.

This indicates that when designers agree with an argument underlying a critic, they tend to strengthen its associated specification condition. When designers disagree with the argument, they tend to disregard the associated specified condition instead of disregarding the argument itself.

Case 2. The subject group was performing the design task for two cooks. They already had constructed a floor plan in CONSTRUCTION, and they wondered if there might be better examples in the catalog. The ordered catalog examples were presented in CATALOGEXPLORER, and they did not like the example at the top of the list.

> **[15]:** *Ummm.. I do not think so* [i.e., the example is good for two cooks]

[the partner asked] *Why not?*
Because here, I can show you. .. Now with two cooks where do they chop vegetables?
[the partner asked] *Aha..*
When they cook, there is no .. one is in front of the refrigerator and one is in the corner .. They are too close to each other.. not enough room. I like our kitchen [a constructed floor plan in CONSTRUCTION] *better especially for two people.*
[the partner asked] *Yeah, for two cooks ...*
Well.. I think this [their kitchen] *is probably better for a single person. Because usually if you are alone, you go for a refrigerator so you walk in this area by yourself, so there is no interference, whereas if you are working with two people, there is interference with each other because one is standing in front of the stove and you have to pass him to get to the refrigerator and get to pass him again. .. so if you have the refrigerator* [pointing to the other side of the kitchen] *here, you can get vegetables from refrigerator, and you can chop while he cooks ... I want to have a stove from the other side of a sink and a refrigerator, so that there is no interference ...*

By looking at a delivered catalog example, the subject found a flaw, or a breakdown, in the example. She explained it to the partner and looked at their current construction again while trying to justify why their current construction was better than the example. Then, she found a similar breakdown in their current construction, which they had not noticed before. Finally, by suggesting how to refine the current construction to the partner, she articulated a heuristics for a kitchen to be good for two cooks.

Subjects seemed to question delivered information more than their own design construction or examples that they had chosen themselves. When the subjects found a breakdown, they often could form heuristics. They often wanted to apply the identified heuristics either to their current construction or their selected example in the search task, or to search for other examples in terms of the identified heuristics. It was often the case that their current construction or their favored example did not satisfy the heuristics, either.

This indicates that designers can reflect more critically on what others (in this case the computer system) have done. The observation corresponds to what Miyake [1986] observed in her study in human-human critiquing. When individuals work on problem solving by themselves, it is difficult for them to evaluate their own solution. By being provided other points of view, the reflection is enhanced.

8.4.2. Response to Delivered Argumentation

Rarely did subjects ignore the delivered argumentation. When they agreed with it, they either applied it to their current design and modified it, or further explored related arguments and sometimes add new arguments. When they disagreed with the argument, they either articulated counter arguments or ignored it.

1. *Agree with a Delivered Argument*

   - *Apply to the Current Design*

     An example of this type of reaction was presented above (see the transcript **[14]**) when the subject agreed with the critic about the type of refrigerator in terms of the microwave specification, and decided to modify the construction.

   - *Identify a New Argument*

[16]: [when the system suggested to have an L-shape kitchen for two cooks and the subject read an argument for the L-shape kitchen] *I see ... Let's check out the argument for an island kitchen and what it said about two cooks kitchen ... No...? Umm.. well, could I create another argument?* [the partner invoked the property sheet to create a new argument] *OK. An island kitchen is good for two cooks because it provides additional counter space ... well.. Two or more (cooks) actually....*

In the transcript, the subject was motivated by the delivery to explore the related argumentation space, and consequently added a new argument.

2. *Disagree with a Delivered Argument*

- *Identify a Counter Argument*

[17]: [a critic fired *"a refrigerator should be close to a door"* so that *"incoming cold food can quickly be stored in the refrigerator"*] *No, because a garage is not next to a kitchen, right? You have to carry groceries anyway.*

[18]: [the same critic fired] *all right ... but this way* [pointing to their current construction], *you can come in and put them on the counter, and put it in the refrigerator. So let's add an argument to this ...*
[the other partner] *Yeah. That makes more sense to me .. logically.. because then you go from the refrigerator to the sink with food and you can go with...*

This type of reaction was most often found. When the subjects disagree with a presented argument, the subjects often articulated arguments against the argument. The refrigerator-door critic rule was disputed by many subjects. The subjects came up with many arguments against the incoming food argument. Some of such counter arguments were generally discussed (i.e., transcript **[17]**), and others were discussed in terms of their current construction (i.e., transcript **[18]**).

- *Ignore*

[19]: [the answer *"you do not need a dishwasher"* is suggested because the subject specified to have a microwave oven, and the subject looked at the related argument] *Because you often use... Oh .. I see. So because the material you put in the microwave you can either throw it away,...umm ... I have to disagree with that.. but that's OK... OK.. so now, but I still think we need a dishwasher.*

The subject disagreed with the presented argument, but did not do any specific response in terms of this disagreement. This type of response was observed occasionally but not too frequently.

8.4.3. Response to Delivered Catalog Examples

The reactions to the delivered catalog examples include that the subjects either discovered favorable features or potentially problematic features. After they discovered such features, they either apply the features to their current construction and specification and modified them, or searched for other catalog examples that had the same features.

1. *Discover a Favorable Feature in a Delivered Catalog Example*

- *Apply to the Current Design*

[20]: [*Dreyer-Kitchen* is delivered. The subject was reading the specification of the example.] *.. both work, seven family .. we do not know how many family she has ... we just don't have enough information about the size of family, and the size might grow, so.... Can I ask you the size of family now??*

The subject was given the requirement description (Figure 8-1), and he had not been aware of the concept of *"family size"* until he read the specification of *Dreyer-Kitchen.* Then, he

realized that the concept is important for the design task and developed the specification after asking the question.

- *Explore the Catalog by Applying it*

  [21]: [an island kitchen was delivered.] *It's so nice to have an open kitchen.. and some people may...*
  [the other partner asked] *What's so good about an open kitchen?*
  OK I mean when you have an open kitchen... umm .... I personally like this because for example, when you have a guest you can still talk to them while you are in the kitchen. On the other hand, some people may think ... umm... while they are preparing very tasty meals, and the whole house smells like a food, and this is not such a good idea. Therefore, there is this trade-off. Personally I am a kitchen person, and I would like to have a kitchen feeling everywhere.. all right, let's look for other open kitchens..

  Until the subject looked at the island kitchen with the open space (with no wall to the next room), he had not articulated the notion of *"open space."* First he liked the idea, then he started to consider it in terms of requirement specifications. He could map the structural feature (i.e., open space) to an abstract functional feature (i.e., smell, kitchen atmosphere, having guests in the kitchen). Eventually, the subject decided to look for other catalog examples that have the same feature.

2. *Discover a Problematic Feature in a Delivered Catalog Example*

   - *Apply to the Current Design*

     An example of this type of response was presented above (see the transcript **[15]**) when the subject did not like the catalog example for two cooks. The subject formed a vague heuristics, applied it to her current construction, and discovered that her design did not satisfy the heuristics either. Then, she more clearly articulated the heuristics in terms of a rule.

   - *Explore the Catalog by Applying it*

     [22]: [looking at a delivered catalog example] *OK... a stove in the corner. You know, when you are left-handed, and then your elbow hits the wall all the time. This is bad.. Umm.. . How about other kitchens.. do they have stoves in the corner?*

     In this script, the subject found a feature (i.e., a stove in the corner of the kitchen) as problematic. Then, he wanted to find other catalog examples that have the same feature, wondering if this feature is very common and his identified feature was problematic or not.

8.5. Problems Found and Short-Range Future Directions

Some of features of KID were articulated as problematic by the subjects. Responses to the observed problems provide a short-term future direction in how to improve KID.

Losing Problem Context. Subjects often asked the experimenter, *"Where was I?"* or *"So what should we do next?"* By following the traces provided by the system's delivery mechanisms, subjects could end up with an unexpected situation, for example, being stranded in CATALOGEXPLORER. It was sometimes difficult for the subjects to go back to the stage when the system started to deliver the information.

Two approaches are considered. First, KID can provide a mechanism that records the global history of user interaction. KIDSPECIFICATION provides the history of attended questions, but it is only

within the context of KIDSPECIFICATION and is insufficient. Designers have to be able to easily put themselves back in the situation where they were in before.

Another possibility is to provide a process guidance in use of KID, such as a checklist. Whereas the checklist provided in FRAMER [Lemke 89] is not task-specific and does not take into account requirement specifications, KID needs to provide a task agenda or other context-sensitive process guidance that takes account the designers' partial construction and specification.

Losing the Origin of Information. Some subjects got confused and said, *"Where does this information come from?"* or *"Where has it gone?"* For example, when one of the subjects was looking at one catalog example in CATALOGEXPLORER, he became interested in adding some answers to the current specification. He went to KIDSPECIFICATION and modified the specification. When he came back to CATALOGEXPLORER, the system automatically reordered the catalog examples according to the new specification, and the one he was looking at was not presented on the display any more.

Delivery of knowledge can be quite confusing. The subjects sometimes felt that the information had "fallen out of the sky" without understanding what was happening. KID provides the underlying rationale as to why the information is delivered, but does not provide meta-level information such as how this information has been delivered by which mechanism based on which part of the task having been done.

Although some of the above phenomena might not happen if the subject were a frequent user, KID needs to provide designers with information about which part of the system would be affected by which transactions the designers make as well as information about bases of the delivered information.

Desire for Partial Automation. After one of the subjects specified that she needs a three-asymmetric-element stove using KIDSPECIFICATION, she started using CONSTRUCTION and complained, *"I answered that I need the stove. Why it did not show up?"*

There is a fine line between how much the system can automate and how much control users have to possess. Although she wished the system to automate the part above, others might have felt the system's automated behavior was too much. Because KID has been designed to be a cooperative assistant with designers, the system will not make any design decisions by itself. However, when there is the deterministic mapping from a specification to a construction, it is possible to automate a small part of the design tasks.

Another approach, which seems more feasible than partly automating design tasks, is to use the constraint-based technique [Gross 90]. As opposed to critiquing mechanisms, in which designers can do whatever they want and the system checks afterward whether what they have done is valid or not, the constraint-based technique will not allow designers to do what violates predefined rules. For example, if designers specify the type of stove in KIDSPECIFICATION, then the designers are not allowed to select other types of stove from the palette in CONSTRUCTION.

Insufficient Retrieval-by-Matching. The expert subject liked the retrieval-by-matching construction mechanism because in kitchen design, designers often need to locate catalog examples that have specific features. For example, when clients want to have the refrigerator near the door in their partial design, designers can show the clients catalog examples each of which has a refrigerator near a door, and give the clients a feeling of *what-if* the refrigerator is near the door, without modifying the partial design.

On the other hand, three limitations of the retrieval-by-matching mechanism of CATALOGEXPLORER were found:

- *Limited scale of construction match.* Identification of types of design units and major configurations that are among prestored conditions was found insufficient for defining a construction match. For example, the subjects could not retrieve catalog examples that have an island without a stove.
- *Lack of retrieval by analogy in construction.* If one of the design units used in the construction did not match any examples, the system retrieved no examples. Even if the two floor plans look very similar, the system cannot deal with global analogy.
- *Lack of mapping from the current specification to the construction of catalog.* Currently, retrieval by mappings is done within the specification and within the construction independently, without any cross referencing. For example, when the subject specified that he needed a double-bowl-sink in KIDSPECIFICATION, the retrieval by matching-specification mechanism retrieved no kitchen examples because none of the catalog examples had the explicit specification about the existence of double-bowl sinks although some of them actually had the type of sink in their constructions.

The current design decision was made because a mismatch may exist between the specification and the construction of a catalog example. For example, a specification for a catalog example says to have a dishwasher, but its construction does not have a dishwasher in it. Therefore, the designer's specification must be matched not just to the specification of the example but to both the construction and the specification of the example. When there is a mismatch between the specification and the construction of a catalog example, the catalog example can provide useful design rationale that describes why the mismatch was made in terms of other factors as constraints. For example, with the aforementioned example that does not have a dishwasher in its construction in spite of the specification, a designer can learn why the example ends up with no dishwasher by exploring related argumentation.

Quality of the Catalog Base. Some of the subjects complained about the poor quality of the catalog base. Currently, the catalog examples have been randomly collected by asking students at the university to design kitchens for themselves. The problem of quality, or the shallowness of the catalog base, or other knowledge bases, is interdependent with how often people use the system. If the system provides rich knowledge bases, people are more willing to use the system. As more people use the system, the knowledge bases of the system become richer through accumulating design knowledge.

Determination of which types of cases should be in the catalog base, is one of the main research topics in the case-based reasoning community. Two extremes are to have only prototypical good designs [Gero 90] or to include interesting failure cases [Kolodner 91]. Another approach would be to collect designs that have a certain *style*. For example, catalog examples designed by the same designer may have a common style, independent of specific problems. The topic should be further studied in order to increase the quality of the catalog base.

Lack of a Comparison Mechanism. The subjects sometimes needed information about how different their current construction was from a retrieved catalog example. The *Evaluate Example* command in CATALOGEXPLORER allows designers to view a catalog example only in terms of their current specification; not their current construction. Also, CASE-DELIVERER, which computes appropriateness values of each catalog example in terms of the current specification, does not compute the value of the current construction.

When using the case-based information provided by the catalog base, the subjects wanted to switch back and forth between an example and their current design. Equal functionality should be provided for evaluating a catalog example and for evaluating the current partial design.

Lack of Global Constraints. It was problematic that KID does not provide a mechanism that allows designers to specify global constraints, such as the shape and area of a floor plan, or cost. Some of the subjects did not like some catalog examples only because *"this is too big for my floor plan."*

In a real life situation, the floor plan and cost are primary constraints that limit the design space quite drastically. Global constraints must be added before KID can be used for real design tasks.

Lack of Accommodation to Specific Needs. One subject group made a statement that they wished KIDSPECIFICATION allowed them to denote specific needs such as having a phone center or having a wine rack, which characterize the uniqueness of their design. Although users of KID can do this using existing tools, the tasks are complicated and require a considerable amount of cognitive effort. For example, for specifying a need for a *"phone-center,"* designers first have to create a new question, answer, and argument about a phone-center using the property-sheets (see Chapter 7). Then they have to create a new design unit, called *"phone-center,"* and may define critic rules for the new design unit. While these tasks are supported by the MODIFIER system [Girgensohn 92], they are still time-consuming and difficult.

KIDSPECIFICATION is designed based on the assumption that a large amount of problem specification is overlapped and used for many design situations. Therefore, although possible, support for accommodating to a new need has not been investigated enough. The problem is a challenge for designing the next generation of a specification component.

8.6. Conclusion

Overall, the study indicated that KID augmented designers' skills in kitchen design, at least those of the novice designers observed in the studies. The results of the survey revealed that all of the novice subjects acknowledged that using KID affected their kitchen design activities. This indicates that they have used, and possibly learned, design knowledge through KID, knowledge that they had not possessed previously. Although the expert responded that the final design product was not affected by using KID, she mentioned that the processes were surely affected; for example, she thought about design issues that she might not have encountered without using KID.

Both the novices and the expert appreciated the knowledge delivery mechanisms of KID. The reasoning behind the delivery was helpful for novice designers for educational purposes, and the reminding and weighting mechanisms were helpful for the expert.

An explicit representation of problem specification provided by KIDSPECIFICATION helped the subjects in understanding the problem better early in the design process. Reading and answering questions given by KIDSPECIFICATION prevented the subjects from overlooking important considerations, and yet did not prevent them from looking at the problem in a new way, for example, *"to have a kitchen feeling everywhere in the house"* (see the transcript [**21**]).

KID supported the seeing-framing-seeing cycle. The seeing, reflecting on their current partial construction and specification, was often triggered by critics and ordered catalog examples. Delivery of sometimes *unexpected* information was an effective way to trigger the subjects to reflect on their task at hand.

Specific critics identified by RULE-DELIVERER were based on their partially framed specification and construction represented in KID. Through the critique messages, the subjects almost always attended to the associated argumentation, which provided a mapping between their current specification and the identified problematic situations in the construction. This triggered the subjects to reflect on their partial construction and specification. Consequently, it either (1) gave them a direction on how to *frame* the partial design by 1-(a) modifying the construction or 1-(b) modifying the specification, or (2) invoked them to articulate new design knowledge, that is, counterarguments to the argumentation, which had been tacit before.

The seeing and framing cycle of processes was also driven by presenting catalog examples. The subjects often discover new features, which are breakdowns or important considerations they had not known before, in catalog examples presented by CASE-DELIVERER. The features spanned both the construction and the specification of an example. For the subjects, it was easier to challenge what others had done than what they had done themselves. Because the judgment of relevance used in delivery is made by KID, the subjects felt free to *critique* the system's judgment. Then, the subjects applied the discovered features to their own design task and often found that their own design had, or lacked, the same features.

The subjects often reacted to delivered knowledge and argued against the delivered knowledge in terms of their task at hand. When being given an object to think with, people start thinking about it and trace associations, which may be linked to a tacit part of design knowledge [Polanyi 66]. Thus, it was easier for the subjects to become able to articulate design knowledge, which had been tacit before, than to start articulating design knowledge, given no context.

Another benefit of knowledge delivery was that it encouraged the subjects to explore the system's information space (see transcripts **[16]** and **[21]**). There is evidence that people search longer for answers to questions when they believe they know the answer [Reder, Ritter 92]. Thus, high *feelings of knowing* correlate with a longer search time. When KID delivered information that was relevant to the task at hand, but not quite right, then they gained this "feeling of knowing," which made their information search longer.

The study also revealed that even if the relevance of delivered knowledge to the subjects' task at hand is quite low, it still supported the subjects in invoking the seeing-framing-seeing cycle by triggering them to reflect on their task.

Finally, the success of knowledge delivery depends on its integration with good access mechanisms. As discussed above, subjects were often motivated to explore related knowledge spaces in KID after having the delivered knowledge. An interesting observation was that subjects complained only about poor retrieval mechanisms (see Chapter 8.5), but not about quality of relevance judgment used in delivery. When the system did services for the subjects with no additional efforts, they appreciated it seemingly regardless of the quality of the delivered information. When the subjects had control and they had a specific task in mind, however, they did not feel comfortable if the system's capability limited what they could do.

8.7. Summary

In summary, the study shows that KIDSPECIFICATION in KID allowed the subjects to better understand their design task. The information provided in KIDSPECIFICATION, in turn, increased the quality of knowledge delivery by having more information about the subjects' task at hand. Not only the delivered knowledge itself, but also the inferred relevance by the system, which provided information about dependencies between the specification and construction, helped the subjects in reflecting on their designs and gave them directions of how to reframe their partial construction and specification. Delivering knowledge also supported the subjects in becoming aware of a part of design knowledge that had been tacit.

Chapter 9

Related Work

The achievement of KID is to integrate the components of the multifaceted architecture. The system integrates a computer-aided construction kit, a hypermedia specification component, an argumentation base, a catalog base, and knowledge delivery and access mechanisms. The integration of KID addresses individual research issues and challenges, such as critics, case-based reasoning, adaptive agents, and reuse of design rationale. In this chapter, I first describe other approaches for building knowledge-based design environments in various design domains. Then, I describe other approaches in each of the four research topics and compare them with the approach KID has taken. Note that the related work discussed in this chapter consists of the research efforts that focus on the topic in particular. KID has not been developed to study the topic per se, but has exploited the techniques in order to provide a design environment that cooperatively supports people in performing design tasks.

9.1. Knowledge-Based Design Environments

Attempts have been made to develop architectures for knowledge-based, integrated design environments. In this section, I describe four representative design environments, two for mechanical engineering, and the others for software design and programming. Although all of these architectures share the same set of functionalities with ours to some extent, what distinguishes our architecture from the others is in its emphasis. Our multifaceted architecture is intended to use AI techniques to enhance designers' "*awareness*" in design, whereas the others use AI techniques to reduce some mundane tedious design tasks by automation. In other words, although our focus is to build *collaborative* systems, their focus is to build *apprentice* systems.

Edinburgh Designer System (EDS) [Smithers et al. 89] uses AI techniques for supporting mechanical engineering design as a knowledge-based exploration task. EDS uses the *exploration-based* model to organize and express its understanding of the underlying knowledge process of design. The model (see Figure 9-1) consists of (1) a Requirement Description, which is equivalent to our specification, (2) a Design Knowledge-base, which consists of knowledge of the domain and knowledge about how to design in the domain, and (3) a Design Description Document, consisting of the requirement description (i.e., completed requirement description), design specification (equivalent to our construction), and history of the design process.

The architecture shares some features with our multifaceted architecture. Having feedback from the *Design Description Document* to the *Design Knowledge-base* supports evolution of the design environment, and having feedback from the *Design Description Document* to the *Requirement Description* supports coevolution of specification and construction. Important differences of their model from the multifaceted architecture are, however, that they put strong emphases on automat-

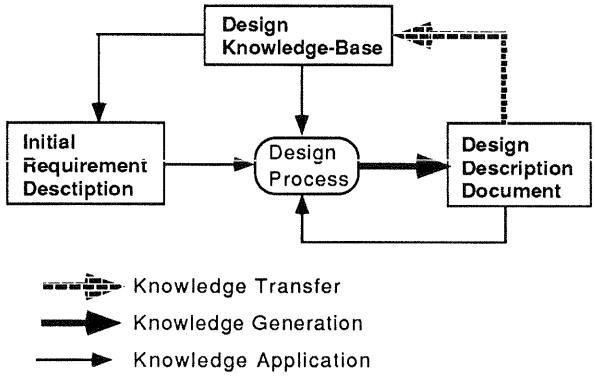


Figure 9-1: Exploration-based Model of EDS [Smithers et al. 89]

---

ing knowledge application processes. That is, their research focuses on formal representations of design knowledge so that the system can automatically refine the *Requirement Description*, and can automatically produce a partial *Design Description* from the *Requirement Description*. Although they do not mean to completely eliminate the roles of human designers by any means, their goal is to decrease the amount of needs for human designers' intervention as little as possible. This approach aims at a totally opposite direction from ours, which tries to empower human designers and support them in doing design.

Silverman and Mezher [1992] described the DSE (Design Support Environment) system and its critics. DSE uses a model for design consisting of a *Generate-Test-Refine-Remember* processes. The architecture of DSE has three layers: (1) an information layer, which provides the databases and models of the domain that permit the machine to generate, test, and remember design elements; (2) a visualization and analyses layer, which includes CAD and block-diagram packages to help generate the design specifications, as well as graphics to display the results of the simulations; and (3) a synthesis layer, which uses a knowledge base. Orthogonal to these layers are iterative processes, which include: generation of the design, testing of the design, and remembering successfully tested designs. Adding the *remember* step transforms the waterfall model into a memory-intensive process that more nearly mimics one prevailing view of the designer's cognition. Although the components of DSE are fairly similar to those of the multifaceted architecture, DSE does not support the ill-structured nature of design. The DSE model presupposes that there is a *stopping rule*, which determines whether a design is complete or not and does not support fluctuating problem specification.

The Knowledge-Based Editor in Emacs (KBEMACS) is implemented to be able to act as a semi-expert assistant to a person who is writing a program, as part of the Programmer's Apprentice project [Waters 85]. The architecture allows KBEMACS to be a *programmer's assistant*, having shared knowledge with the programmer while both of them interact with a programming environment, including editors, compilers, and debuggers. The system has a library of programming

cliches, which are standard methods for dealing with a task — lemmas or partial solutions. The cliches provide the basic vocabulary used in communication between man and machine and embody most of the knowledge shared between the programmer and the system. Finally, the system has *plans*, which provide simple reasoning about programs and combine cliches together to create programs.

The overall goal of KBEMACS is to make expert programmers more productive. In this sense, their approach shares a similar goal with ours. However, the difference is that KBEMACS does not view programming as an ill-defined design task. The concrete goal of KBEMACS and its underlying Programmer's Apprentice project is to automate a tedious mundane part of the programming tasks so that human programmers are relieved of such tasks, rather than to support designers who enjoy designing. Their focus of the approach is to identify automatically which part of the task can be delegated to the computer assistant.

The IDEA system [Lubars 89] is a software design environment that relieves software designers from the more mundane and mechanical aspects of design by automation, and thus it permits them to devote their energies to the more difficult and intellectually challenging aspects of design. The architecture integrates seven aspects of software development activities: clerical support, analysis support, testing support, organizational support, interface support, knowledge-based support, and intelligent support.

IDEA is based on a dataflow-oriented notion of software design, and the system automatically balances various levels of hierarchical dataflow diagrams. The system maintains a goal agenda to track a variety of design goals, interfaces with additional tools to provide prototyping capability, provides support for accessing a knowledge-base of reusable design abstractions and design refinements, and provides several forms of intelligent design assistance, such as simple constraint propagation and planning. Although IDEA tries to automate generation of a design artifact based on a formal functional specification, both their approach and ours have a principal goal in common — to empower human designers by providing integrated design environments. The interesting feature of IDEA is that the architecture incorporates social aspects of design activities such as organizational and clerical activities. Lubars's approach [1989] started from viewing software design as a social activity, and using AI techniques to relieve designers of routine tasks, instead giving them more opportunity to enjoy the creative intellectual parts of design activities.

9.2. Computational Critiques

Most existing computational critiquing systems support only *one-shot dialog*. As observed in the user studies, when a critic was fired, the subjects often disagreed with it and wanted to argue against the argument underlying the critic. Critiquing is a process, not a one-time event; a critiquing mechanism has to accommodate to users' responses and corrections to the critics [Silverman, Mezher 92]. The critiquing mechanism of KID gives designers access to related arguments, and KIDSPECIFICATION allows designers to augment arguments in response to the accessed arguments.

The newly added argument is used to dynamically derive a new specific critic rule using specification-linking rules. Thus, critics in KID partially address the problem of knowledge acquisition.

A critiquing mechanism in FRAMER II [Lemke 89] allows users to have more control over its behavior. The system supports users in designing an interface display on SYMBOLICS LISP machines. The rules evaluate the completeness and syntactic correctness of the design as well as its consistency with the interface style used on SYMBOLICS LISP machines. Each critic is classified as either mandatory or optional. Each of the fired critic messages displayed is accompanied by up to three buttons that provide options: *explain*, which displays an explanation about the reasoning a designer should consider and about how to achieve the desired effect; *reject*, which disables the critic rule so that it will not fire again; and *execute*, which automatically performs a reasonable default solution.

The critiquing mechanism in KID provides the explanation facility by presenting a link to the argumentation base. The information in the argumentation base is not just an explanation, but provides deeper reasoning for the identified problematic situation. KID does not provide either reject or execute facilities. The execute facility is related to the issue of partial automation discussed in Section 8.5. The rejection facility addresses the issue of how to deal with intrusiveness of critics. However, this approach is problematic because a critic that has once been rejected remains disabled unless users enable it; critics that were considered irrelevant may become relevant as design situations change, and designers may not become aware of this change. Instead of providing a control to reject irrelevant critics, KID provides specific critics identified by RULE-DELIVERER, which takes into account the designers' problem situation, and therefore, decreases the chance of firing irrelevant critics.

The DSE system mentioned in the previous section provides critics [Silverman, Mezher 92]. In its CAD environment, CLEER, Silverman and Mezher [1992] observed that the typical designer is unlikely to absorb or even read praise information, which corresponds to the finding in the present study, presented in Section 8.3.7. They identified a need for a prototype analogy module, which would help users figure out the next design move to make. KID provides the catalog base, which is a source of such prototypes. CASE-DELIVERER uses the specification-linking rules to help designers make analogies between their problem specification and the catalog examples.

Silverman and Mezher [1992] identified three categories of critic strategies, including: (1) *influencers* strategies for preventing errors, (2) *debiasers* for identifying possible breakdowns in the current design, and (3) *directors* for assisting users in organizing design tasks, such as process guidance, checklist, or prescriptive processes. In KID, delivered catalog information can be viewed as an *influencer*, and specific critics are *debiasers*. The need for *directors* is identified in the study as discussed in Section 8.5.

In DSE, there is no notion of "specification," or no mechanism that allows designers to specify

their goals or objectives. For effective criticism to occur, there must be a mutual, two-way exchange of ideas; KIDSPECIFICATION in KID provides the necessary shared knowledge between the knowledge delivery mechanisms and the designers.

In software design, Fickas and Nagarajan [1988] studied how specifications evolve from an often vague set of client descriptions to a formal specification. Their KATE system supports domain analysts and end-users to develop formal specifications. The system gives "policies" as abstract requirements, maps them to concrete cases, supports detecting conflict among policies by critiquing the design specification, and generates scenarios. Mapping to the terminology used in this dissertation, their *"policy"* is covered by KIDSPECIFICATION, and their *"formal specification"* conceptually corresponds to what CONSTRUCTION offers. *"Critics"* provided by KATE check for compliance between policies and formal specifications.

KATE allows users to assign three types of expressions to policies to assist in making trade-offs among them: *important*, *unimportant*, and *unknown*. In KIDSPECIFICATION, designers can represent relative importance using a scale between 1 to 10, which is more flexible than using the three discrete values.

Critics in KATE identify three types of breakdowns: (1) nonsupport (stated as an important policy but not present in the formal specification), (2) obstruction (an important policy conflicts with what is presented in the formal specification), and (3) superfluousness (an unimportant policy, but present in the formal specification). In KID, specific critics check nonsupport and obstruction types of breakdowns, but cannot check for superfluous breakdowns. In KIDSPECIFICATION, designers cannot distinguish what they do not care about from what they want to disregard; both are done by not selecting an answer. In kitchen design, there has been no need for disregarding specific requirements, but different domains may require the mechanism.

Problems of the KATE approach include that no further explanation is provided as to why a policy supports or obstructs a certain part of specification, and that an issue-policy dependency cannot be modified by end-users. Their model of specification is very limited, whereas in KID, specification-linking rules, which represent dependencies among specification (i.e., policy) and construction (i.e., formal specification) are linked to the argumentation base, which provides explanations underlying the dependency. Furthermore, questions, answers, and arguments in KIDSPECIFICATION can be added by end-users. The model of KIDSPECIFICATION does not limit the space of designers' specification.

9.3. Use of Catalog Examples: Case-Based Reasoning

Catalog examples provide a source for case-based information. Activities involved in case-based reasoning (CBR) are (1) retrieval of cases, and (2) adaptation of cases to the problem at hand. KID supports retrieval of catalog examples, but does not provide automatic adaptation of the examples. Instead, KID provides mechanisms on how to adapt the catalog examples by allowing designers to

evaluate the examples in terms of their problem (i.e., the *Evaluate from Current Specification* command; see Section 7.1.3).

None of the existing approaches in CBR, except for Jakiela's [1988] system described below, provide *delivery* of cases, which requires a system to take the initiative in case retrieval and automatically retrieve cases relevant to the users' task at hand. This reflects the fact that only few systems have combined a case-base with tools for construction and specification. CBR started as an automatic problem-solving paradigm that excluded human intervention. Although its focus has shifted to case-based decision-aiding systems [Kolodner 91], in which people play the crucial role, many CBR systems are still stand-alone and separated from humans problem-solving activities. KID, in contrast, is one of the few systems that actually aims at being a case-based decision-*aiding* system — by integrating tools that support problem specification and solution construction with a case-based reasoning paradigm.

Jakiela [1988] has developed a system similar to CASE-DELIVERER. Jakiela's suggestion-making CAD system helps users obtain better design ideas and brings information normally not considered during preliminary design to the initial design stage. The user builds a design with a predefined set of geometric features, and the system makes suggestions by altering the design while preserving the same set of features. The system, however, simply lists all the possible suggestions without taking into account the appropriateness of each suggestion. Jakiela's user study reported that users sometimes followed nonoptimal design suggestions when better suggestions were available and possible because the users were not willing to browse all the suggestions. This observation indicates that suggestions should be presented in rank order.

CASE-DELIVERER in KID has gone beyond the suggestion-making CAD system in several ways. First, suggested catalog examples are *ordered* by CASE-DELIVERER according to the computed relative importance, thereby circumventing the above problem. Second, the rules that make suggestions (i.e., specification-linking rules) are dynamically derived from the argumentation base. Thus, the rules are accommodated with an update of the knowledge base. Third, KID can provide designers with an explanation as retrieval rationale about why the suggestion (ordered catalog examples) has been made by CASE-DELIVERER in the way it has. Because the rules are derived from the argumentation base, the original arguments provide explanations for the judged relevance.

Problems with current case-retrieval approaches are discussed by Wolverton and Hayes-Roth [1991], including (1) the importance of taking into account the problem-solving goal and retrieving useful cases for the goal instead of retrieving structurally similar cases, (2) difficulty to anticipate what the salient features of a case are at case storage time, and (3) a need for users to have control over defining "match." These situations are very problematic, especially in the design domain. As discussed in Section 7.2.3, we cannot make assumptions that certain clearly delineated parts of knowledge can be determined a priori to be worth recalling later.

KID addresses all three issues by having a specification component. First, KIDSPECIFICATION

provides the explicit representation of designers' problem-solving goal. Second, instead of classifying cases at storage time by defining salient features a priori, the specification-linking rules are used to perform analogical matching to the users' task at hand. Third, the specification-linking rules are dynamically derived from the argumentation base. When designers modify the argumentation base, the rules are also recomputed. Moreover, the rules are weighted according to the relative importance, or weights, that designers associate with selected answers. Thus, designers have more control over the retrieval.

Mark and Schlossberg [1990] supported case retrieval by combining specified constraints, rules, and browsing. Instead of indexing designs and classifying them as cases, they store design as a whole as a ''design memory'' and use *constraints* to eliminate irrelevant information. As designers make decisions, starting with the initial specification (i.e., design constraints), they are presented with the part of the design memory that is compatible with their current design. As further decisions rule out more of the structure, the system will zoom into a narrower space in the memory, eventually becoming a "browser" for the network at the individual decision level. Then designers can begin the step-by-step examination of the memory that they call *'' navigation.''*

Similarly, KID collects all designs as raw design memory without classifying them. CASE-DELIVERER uses constraints given by KIDSPECIFICATION to help designers to *''zoom in''* the related space. CASE-DELIVERER does not eliminate irrelevant information, but instead orders the catalog examples. Their system does not provide a link to a design construction. The information given as constraints is used only for searching the design memory. In contrast, KID uses this information also for keeping consistency with the information in CONSTRUCTION with specific critics.

CYCLOPS [Navinchandra 88] uses a type of analogical matching, called *systematicity-based match*, which finds common causal relationships among attributes in the problem at hand and cases rather than just common attributes. The system supports designers in (a) performing goal-directed reasoning (b) managing goals and dependencies among them using a truth-maintenance-like framework, and (c) building explanations of solutions to cases as it solves them, which allows designers to reuse the solution as a new case.

CYCLOPS stores a causal explanation of the goals and subgoals of the case with each case. The matching is not based on surface features of stored cases and the task at hand but on a deeper understanding of why they are so. They use a demand-posting method, which is self-interrogation while redefining the problem question by finding and addressing the causes and effects of the problem.

The demand-posting method is similar to the use of KIDSPECIFICATION and the specification-linking rules. In CYCLOPS, demand-posting is formal, and systematic. The explanations stored in the cases are predetermined and static. Predetermining an explanation of case information also predetermines the ways in which the case can be reused, which fixes the view of the case. They want to organize and index cases according to the associated explanation, but this is particularly

difficult for a design domain because the explanations are not fixed but change in response to the purpose for which the cases are being retrieved. Instead of providing predetermined cause-effect relationships, KID uses the specification-linking rules to dynamically derive the explanations. The rules that dynamically determine the purpose of a design, and the *Show Delivery Rationale* commands provides links to the argumentation base as explanations.

CHEF [Hammond 90] is a case-based planner (CBP) for planning recipes. The system uses an approach that is very similar to KID in terms of case-retrieval. In storing a plan in memory, a CBP does not classify and generalize the plan itself. Instead, it generalizes the features used to index the plan in memory. Although a CBP does not need to generalize the plans it stores, it does need to generalize the features used to index them. Because of this, the plan can be suggested for use in a wide range of situations and can still retain the specific information that makes it more useful when applied to situations in which the goals that are being planned for are completely satisfied by the plan itself. By generalizing the indexes rather than the plan, a CBP is able to avoid many of the trade-offs between generality and power of application. The general indexes make it applicable in many situations, and the specificity of the plan makes it a powerful tool in those situations in which the match between the current goals and those satisfied by the plan is a good one.

In KID, CASE-DELIVERER uses specification-linking rules that represent cause-effect relationships for cases. The rules provide dependencies among features of construction and specification. Description is stored in the argumentation base, to which a specification-linking rule provides a link because the rules are derived from the argumentation base. This is very similar to the method that CHEF uses for failure detections. In CHEF, failures were indexed by the features predicting them. Description of failure in CHEF is used to figure out the features that will later predict similar failures. When multiple features are required to predict a failure, all of them are linked to the memory of the failure. This memory is not activated unless all of the linked features are present.

In CHEF, the actual implementation of its memory of plans is via a Discrimination Net in which plans are indexed by the goals they satisfy and the problems they avoid. These goals and problems are linearly ordered by their importance, with the higher priority features used at the higher levels of discrimination in *value hierarchy*. CASE-DELIVERER does not eliminate irrelevant cases, but instead computes values and orders them. In CHEF, these orders are fixed; users cannot change the relative importance of problem goals, as they can in KIDSPECIFICATION, with which designers can modify the problem specification and associated weights at any time.

A spreading activation model is an alternative to using rules for implementing analogical matching. For the spreading activation model, information provided via a specification component also plays a crucial role. Wolverton and Hayes-Roth [1991] presented a goal-driven spreading activation model. The architecture first retrieves a potentially analogous base concept, and then searches through all stored abstractions of that potential base to find one that is also an abstraction of the target. Their system would use the system's knowledge (1) about the design domain, (2) about the generic task of design, and (3) about analogy for retrieval of analogies. The system uses the aug-

mented spreading activation model. The model starts with an initial design goal, which causes some concepts to be assigned activation, and this assignment begins the spread of activation in memory. The spread of activation is guided by the *control* plan. The *matching* component identifies promising concepts from the ones that have been activated, and uses this information to help create a new control plan for the spread of activation.

ARCHIE-II [Domeshek, Kolodner 92] is a case browser as a design assistant in the domain of architectural design. A case in ARCHIE-II provides three types of representation: stories, guidelines, and documentation. *Stories* are selective representation about some particular design case that has some lesson to teach. *Point-stories* explain the interpretation of design features with respect to a single design goal or plan, and *interaction-stories* discuss how some features of a design case can be interpreted with respect to several design goals and plans with possible conflicts. *Guidelines* are the abstractions behind story lessons; design guidelines generalize across cases and provide a way of relating parts of cases to one another, varying requirements; others describe policies or simple recommendations. *Documentation* includes summaries that provide an overview of the entire design, detailed specification for individual parts, the original requirements of the design, evolving requirements, and the eventual design solution that resulted.

In KID, specification-linking rules link catalog examples to arguments in the argumentation base, which provide *point-stories*. Once a related argument is located, the neighborhood pro and con arguments provide *interaction-stories* for designers. The argumentation base provides the *guidelines* for catalog examples. They organize guidelines by starting with a goal/plan hierarchy, rooted at the life cycle from the perspective of particular building-user populations. KID uses the PHI structure for organizing the argumentation base, which has the advantage of being able to utilize serve-relationships [McCall 91]. Finally, the information collected in CONSTRUCTION provides the *documentation*. KID uses a partial specification and construction to restructure the catalog base according to the task at hand in the same way as ARCHIE-II uses *documentation* to guide the case browsing processes; the difference is that KID orders catalog examples with assigned numbers indicating relative relevance whereas the judgment of relevance in ARCHIE-II is binary: relevant (i.e., retrieved) or irrelevant (i.e., not retrieved).

In ARCHIE-II, representation of a case includes: (1) *Design Issue* (e.g., privacy, safety); (2) *Structural Components* (e.g., design unit); (3) *Functional System* (e.g., may refer to either a generic system, such as plumbing or electrical, or a specific component from such a system, such as pipes, value, switch, etc.); (4) *Stakeholders Perspective* (e.g., various classes of individuals and institutions that have concerns about the building, such as designers, builders, owners, users, etc.); (5) *Lifecycle Phase* (during which these stakeholders are likely to have particular goals for the building, such as construction, user, maintenance, and renovation); and (6) *Outcome* (e.g., positive or negative).

In KID, the specification of a catalog example provides *design issues* and the construction provides *structural components*. Although other information can be captured in the specification, KID does

not distinguish types of information in detail. The need for *perspectives*, for example, was identified in the study discussed in Section 8.2.5.

Finally, ARCHIE-II provides *relationship indexes* that link stories, guidelines, and documents. These indexes are hard-coded in cases, whereas in KID, specification-linking rules that serve the same purpose are dynamically derived.

9.4. Adaptive Agents: Information Delivery

As discussed in Section 3.4, information delivery is a type of adaptive system. Oppermann [1992] provided a requirement for an adaptive system consisting of three parts. First, the *"afferential"* component observes and records the user behavior and system reactions on a meta-level. In KID, CONSTRUCTION and KIDSPECIFICATION provide this level. Second, the *"inferential"* component, which is most crucial, analyzes the data gathered to draw conclusions, that is, to identify from the user's behavior possible indicators for adaptation. In KID, the specification-linking rules used by CASE-DELIVERER and RULE-DELIVERER constitute this component. Finally, the *"efferential"* component provides feedback for modifying the system's behavior. The modification can have two different target objects: the application itself or the error and help system. In KID, the ordering of presented catalog examples is modified according to the task at hand, and critic messages are displayed.

FLEXCEL (Flexible EXCEL) [Oppermann 92] is a tool to adapt the system on the initiative of the user, and also is auto-adaptive with the system, taking the adaptation initiative. The user may select from offered suggestions, including (1) working autonomously without having to wait for or being dependent on the adaptive component, or (2) having the authority to reflect on, to accept, or to reject the system-initiated adaptation suggestion. FLEXCEL's menu-like *"adaptation tool bar"* allows users to have control over the system's adaptivity. It provides: (1) a *tip*, which is similar to a critic message; (2) a *tip list*, to which users can stack the tips; (3) *adaptation*, that automatically executes the suggested tip; (4) an *overview* of completed adaptation; (5) *critics* for analysis; and (6) a *tutorial*, which educates users about the adaptation.

Adaptation of the FLEXCEL system fundamentally differs from that of the knowledge delivery mechanisms in KID because whereas FLEXCEL adapts the system's behavior to users, KID adapts only its information spaces. However, the conclusion from the FLEXCEL user study, which states that the evaluation of adaptive and adaptable concepts is most promising when both features are linked to cooperate, is still applicable to KID. KID provides both access and delivery mechanisms. Starting from the delivered design knowledge, designers can start exploring the knowledge-base using the access mechanism, which was actually frequently observed in the user study (see Chapter 8).

Along the same line, the INVISION system [Kass, Stadnyk 92] also uses both adaptive and adaptable aspects. The system uses both "implicit" (i.e., inferred by the system) and "explicit" (i.e.,

input by a user) user models to improve organizational communication, that is, *"who should I tell, and who should I ask"* problems. The system can implicitly build user models by inferring users' information needs based on observations of their interactions with database systems. This is required because not all users will edit their user models and the model they specify will be inaccurate as the usage situation changes.

Delivery is similar to a notion of advice-giving. The advice-giving approach [Carroll, McKendree 87] has been explored in the context of tutoring systems, whose purpose is to educate users. Current work on advice-giving systems has focused on system-initiated advice-giving in the context of user error.

OBJECTLENS [Lai et al. 88] provides a knowledge-based environment for developing cooperative work applications, with a template-based interface that integrates hypertext, object-oriented databases, electronic messaging, and rule-based intelligent agents. A knowledge delivery aspect of OBJECTLENS is that rule-based agents process information automatically on behalf of their users: when an agent is triggered, it applies a set of rules to a specified collection of objects. Rules consist of *"if"* descriptions (i.e., criteria) and *"then"* descriptions. When an object satisfies the *if* part specified in a rule, the rule performs the *then* part. Specification of a rule can be done with provided templates by simply filling in a form. The system provides a simple way to perform database queries; users can simply create agents that scan the objects in one folder and insert links to selected objects into another folder. The rules in the agents specify the criteria for selecting objects. The system provides this semi-autonomous retrieval of semi-structured objects by intelligent agents.

The main purpose of information delivery of OBJECTLENS is different from that of KID. The purpose in KID is to deliver design knowledge so that a designer can use it for solving design tasks, whereas for OBJECTLENS the purpose is to locate objects in the information store. Users of OBJECTLENS have to specify criteria in order to locate the information. With KID, in contrast, the purpose of fired specific critics in CONSTRUCTION and suggestions made in KIDSPECIFICATION are more like notification. Specification-linking rules are derived from the argumentation, which is accumulated in response to a breakdown that designers encounter; designers do not have to explicitly create criteria for information delivery.

9.5. Reusable Design Rationale

Deriving specification-linking rules from the argumentation base is a mechanism of reusing design rationale. Although recording design rationale, which is a description about why and how a design decision has been made, has been found very effective in improving design processes, none of the systematic approaches have been too successful, mainly because of the cost-benefit dilemma [AAAI Workshop 92]. One of the problems is that it is very costly to access the recorded design rationale once it is captured [Yakemovic, Conklin 90; Fischer et al. 91b].

Recorded design rationales by systems such as gIBIS [Conklin, Begeman 88], SIBYL [Lee 90], and DesignRationale [MacLean, Young, Moran 89] are isolated from constructed solutions. Consequently, designers cannot access recorded design rationale through constructed design objects. Recorded design rationale in PHIDIAS [McCall et al. 90] is linked to design components constituting a design solution, but is not linked to the design as a whole.

An approach to addressing the above issue is to integrate design rationale with the design artifact [Fischer et al. 91b]. JANUS integrated the design construction with argumentation. Thus, designers can access the recorded argumentation through a constructed artifact. Another approach is to formalize the design rationale so that the system can manipulate the content of the design rationale. KID has employed both approaches; it integrates design rationale with a solution construction tool and a problem specification tool, and automatically derives specification-linking rules from the argumentation base.

Research has been done in pursuing how to increase formality so that the system can "understand" the content, which is often informally stated in natural language. Lee [Lee 92] described a mechanism that parses the content, identifies domain distinctions, and automatically derives links to other parts of design rationale. In my approach, the system asks users to associate argumentation to predefined domain distinctions (see property sheets in Figures 6-6 and 6-8). An intermediate approach is taken by Shipman [Shipman 93], in which the system identifies possible related domain distinctions and provides them to designers as defaults that designers can subsequently modify. Currently, KID has approximately forty names for issues and a hundred names for associated answers that constitute domain distinctions. As the system grows, this type of support for locating and understanding related domain-distinctions is necessary.

In KID, a single argumentation base is used as a shared memory. Designers continuously use the argumentation base while gradually accumulating design rationale through a number of design tasks. Design rationale has been regarded as useful only for the specific design task. The approach KID has taken is that of regarding the design rationale in the argumentation base as a type of *case*. By collecting specific design rationale in a single argumentation base, generic design knowledge will emerge by being used while refining and tuning.

A problem with this approach is that such an argumentation base can grow quickly and may contain a large number of conflicting views and opinions about design. We need viewing mechanisms to organize and filter the content. One way to deal with this problem is to introduce the notion of *perspectives*. Stahl [1993] discussed supporting design as interpretation, and providing perspectives as an inheritance network. For example, as discussed in the observation (see Section 8.2.5), there can be a perspective for Japanese cuisine, which can be a subsidiary of the *seafood cuisine* and the *rice cuisine* perspectives. Another approach is to organize the argumentation base by authors' names. In KID, when a designer adds a new argument, KIDSPECIFICATION automatically associates the argument with the user name. Later, designers can look at arguments produced by a certain designer and may find a consistent theme in the arguments. As discussed in Section 8.2.5, this

captures a *style of design* if both the argumentation and the catalog examples can be accessed through a designers' name.

In domains such as computer network design, the design takes a long period of time and is performed by a number of different designers. In such domains, it is important to capture how the design has evolved. Certain design decisions depend on others, and keeping the history of the evolution supports designers in understanding their partial design [Reeves 93].

Finally, another unique feature of the approaches of KID is that it allows designers to record design rationale about catalog examples, which are already completed designs. In real life, it is not uncommon to provide *post-hoc rationality*, which means that we can articulate why we have done so after the problem is solved. This *historical revisionism* is often observed in everyday problem-solving activity, such as in a geometry proof [Schoen 92]. Providing design rationale identified afterward is as useful as design rationale captured during the design time in order to help other designers to understand the design, and designers have to be able to record this.

Chapter 10

Future Directions: Applicability and Extensibility of the Approach to Software Development

In this dissertation, I described the KID design environment for architectural kitchen floor plans. A study of the system has shown the usefulness and potential benefits of the underlying multifaceted architecture. The kitchen domain has been chosen as an *object-to-think-with* for the purpose of illustration. The broad-based familiarity of this domain has helped in focusing on the essential issues of the approach without being distracted by the semantics of the domain itself.

The domain-oriented design environment approach has been applied to other design domains, such as user-interface design [Lemke 89], computer network design [Fischer et al. 92b], lunar habitat design [Stahl 93], and phone-based voice dialog design [Repenning, Sumner 92]. Although the approach has been shown to be applicable to a variety of other design domains, some of design decisions made for the KID system have been affected by characteristics of the kitchen design domain including:

- dependence on a spatial metaphor. Most of critic rules in CONSTRUCTION use relations between design units such as *"close to," "away from,"* or *"next to"* to evaluate a design.
- the lack of *enactment.* Completed design constructions in KID can be implemented only outside of the KID environment. Usage situations of designed artifacts are not supported in KID.

Some design domains do not use the spatial metaphor, but techniques for manipulating the spatial metaphor can be used to deal with most relations, for example, to represent temporal relations or relations in data flow diagrams.

The lack of enactment is a more serious shortcoming. In software design, a completed design artifact can be implemented and executed within a design environment; for example, the user-interface design in FRAMER [Lemke 89]. In mechanical engineering, simulation of a design plays a crucial role in evaluating the design. Although simulation of a kitchen design is technically possible to some extent, for example, by using *Virtual Reality* techniques to simulate usage situations, it has been beyond the concern of this dissertation.

In this chapter, I discuss future directions of my dissertation work in terms of applicability and extensibility of the KID techniques to the domain of software design. I first briefly discuss the problems and challenges that software engineering faces. Object-oriented methodologies have been developed to address them and have been found to be useful, but not sufficient [Fischer et al. 93b]. I argue that the multifaceted architecture approach goes beyond the current object-oriented technologies. After discussing how the architecture addresses the problems of software engineering in general, I argue that the multifaceted architecture provides a reuse-based object-oriented design

environment that supports the location-comprehension-modification cycle, and I briefly describe how the components and mechanisms of the architecture can be built in such a design environment. Finally, I discuss challenges and further refinement of the architecture for accommodating to software development.

10.1. Problems in Current Software Engineering Practice

Among the many problems software engineering has been coping with are [Curtis, Krasner, Iscoe 88] (1) the thin spread of application domain knowledge, (2) fluctuating and conflicting requirements, and (3) communication and coordination breakdowns.

The thin spread of application domain knowledge. Software developers have to bridge the great transformation distance between high-level abstract domain concepts and a generic programming language substrate. Requirements of software come from end-users, who at best can describe their problems using their domain language represented in a *situation model* [Fischer, Henninger, Redmiles 91b]. Software developers have to be able to understand what end-users mean and to map it to a *system model* that a computer system can manipulate. The problem of the thin spread of application domain knowledge has become more serious in the last several years as computer resources have become more distributed and decentralized. When they are centralized on mainframe computers, software projects tend to be long-term and large-scale. Those projects typically have a few project members who have learned about the domain and help the other project members in understanding the domain. When decentralized among workstations and fast microcomputers, however, end-users start to expect software to be more flexible and customized. The size of projects have started to scale down, and projects are scattered. Each project wants to have a developer who is familiar with the domain, but it is too costly for a single small project to have a member learn the domain.

Fluctuating and conflicting requirements. Although the waterfall-type development model is still widespread, mainly due to managerial concerns, there has been a shift of emphasis from ''downstream'' activities (transformation of formal specifications into implementations) to ''upstream'' activities (development of specifications in order to decide precisely what to build) [Belady 85]. People have started to experience ''design disasters,'' in which the implementation is correct with respect to a specification that is wrong. Not only theoretical foundations of the ill-structured nature of design, but also empirical evidence has shown that it is impossible to have complete specifications because requirements fluctuate over time and conflict with each other [Curtis, Krasner, Iscoe 88].

Communication and coordination breakdowns. In addition to communication breakdowns between end-users and software developers caused by using different ''languages'' [Greenbaum, Kyng 91], and coordination problems between project members that many CSCW (Computer-Supported Cooperative Work) research addresses [Greif 88], a problem of supporting long-term communication among software developers over time is crucial [Reeves 93]. Empirical analyses

have shown that more than half of the effort in the development of complex software goes into maintenance, and that 75 percent of maintenance effort goes into enhancements, such as changing systems to meet additional requirements [CSTB 90]. In order to support the maintenance, software developers need to understand why the software is built the way it is now. The source of information for this is limited to documentation or information gained by asking the original developers. Often such documentation is insufficient in detail, or the original developers may not belong to the same organization anymore or may not remember what they did. Recording design rationale has not been too successful because of the inherent cost-benefit discrepancy [Grudin 88]. Unless people see immediate benefits from what they do, they will not record design rationale, which requires additional work.

These problems can be attributed to current software development practice, such as the use of a generic programming language; a waterfall type of development model; and the dependence on individual tools such as editors, debuggers, and compilers, each of which supports only a very limited aspect of software development. Moreover, traditional approaches pay little attention to *design* and *designers*. They do not take into account that software development is a *design*, which is inherently ill-structured and open-ended (see Chapter 2). The approaches are *object-centered*, because their methods and tools are organized in terms of software objects to be developed, and not *human-centered*.

10.2. Object-Oriented Technologies

Object-oriented methodologies address the above problems. The object-oriented programming paradigm [Meyer 88] allows software developers to deal with the ill-structured nature of software design (see Chapter 2). Software engineers can start by defining abstract objects [Liskov, Zilles 74] that represent an application domain and gradually refine these objects as they gain the understanding of the problem [Cox 86]. With procedural or functional programming languages, software designers have to know *how* and *what* to implement with clearly defined functionality, which must be transformed from representations of an application domain to representations of the programming concepts. With object-oriented programming languages, the functionality can emerge through iterative refinement of objects as partially defined objects are evaluated and verified.

However, decades of object-oriented methodologies have shown the approach to be necessary but not sufficient [Fischer et al. 93b]. First, support provided by generic object-oriented programming environments is not enough. Existing object-oriented CASE (Computer-Aided Software Engineering) tools that support object-oriented design methodology provide only *domain-independent construction kits* (see Section 4.1). A great deal of effort is still required for software developers to map their own problem domain concepts to representations of generic programming terms. Tasks such as identifying domain abstracts and constructing a library of reusable abstract objects are all imposed on software developers, who also have to build application software. Second, the problem of information overload, described in Chapter 3, holds for class libraries that store constructed objects. The spaces of the libraries can be very large and complex, and deep understanding of the

libraries is required to make full use of the existing objects. Finally, the object-oriented approach does not address the problems of communication and coordination breakdowns. Many CASE tools are still built based on specification-driven development models, in which problem specification is completely detached from solution construction.

10.3. The Multifaceted Architecture Goes Beyond Object-Orientation

The multifaceted architecture (Figure 4-3) demonstrated in the context of KID addresses each of these problems by going beyond the object-oriented design environment [Fischer et al. 93b]. Specifically,

1. Domain orientation in the design environment provides human problem-domain communication. Design units provided in a palette of a construction component are commonly used constructed objects for the domain. Designers evolve design from those domain abstractions. The catalog base provides designers with useful examples that help them to understand how to use the abstract objects.

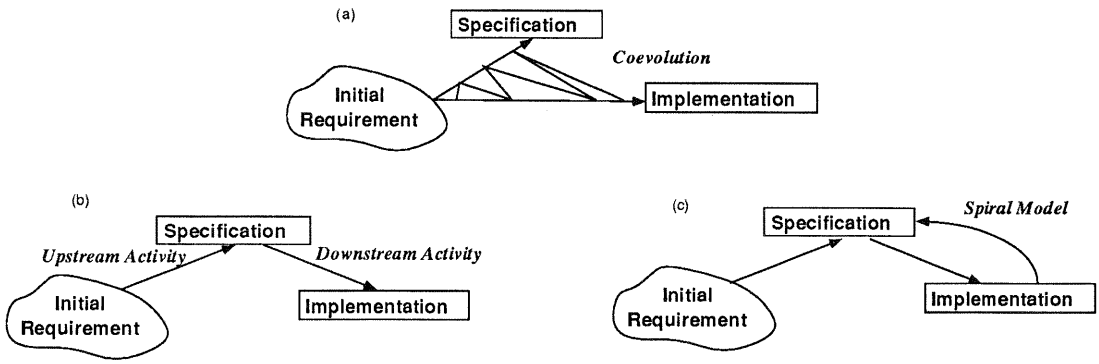
2. Support of the coevolution of problem specification and solution construction, the main theme of this dissertation, addresses the problem of fluctuating and conflicting requirements. Embedded knowledge delivery mechanisms support designers in reframing specification and implementation.

3. The knowledge bases of the multifaceted architecture provide media that facilitate long-term indirect communication among designers. Additionally, a specification component and other components can be design media that facilitate communication between end-users and designers.

Many object-oriented CASE (Computer-Aided Software Engineering) tools provide construction components of design environments [Booch 91; Meyer 88]. Partial constructions (implementations) constructed in the CASE tools serve as prototype systems. Class libraries of reusable abstract classes provide a palette of a construction component. Example application programs use those components from a catalog base. Design rationale recording systems such as gIBIS [Conklin, Begeman 88] provide an argumentation base. A specification component can be built on top of the argumentation base, but may have a domain-specific user interface design, for example, a spread sheet form instead of the questionnaire form that KIDSPECIFICATION currently provides.

Applying the multifaceted architecture to object-oriented software design suggests integration of those existing tools and techniques. If the argumentation base is based on a variation of the IBIS structure, specification-linking rules can be established and the same inference engine can be used to maintain the interdependency between specification and construction. Thus, such integration enables knowledge delivery mechanisms that help designers locate, understand, and modify objects.

Domain-Orientation. KID is a type of domain-oriented, object-oriented design environment. Design units in the palette in CONSTRUCTION and catalog examples in the catalog base are reusable object classes. Palette items provide abstract classes that software developers will use and modify



(a) illustrates coevolution of problem specification and solution implementation in the integrated model. Redefining of initial requirements is reflected into problem specifications. (b) illustrates the separation of upstream and downstream activities. (c) illustrates the spiral model.

Figure 10-1: A Model for Coevolution of Specification and Implementation

---

in order to produce a new application in the domain that accommodates the task at hand. Catalog examples provide concrete classes as examples to illustrate how the abstract classes can be used and applied.

Coevolution of Specification and Implementation. As discussed in Section 4.4, the multifaceted architecture supports coevolution of specification and construction.

Figure 10-1 illustrates our model of coevolution of specification and implementation. The term *implementation* is used for construction of software design. As discussed in Section 4.4, this notion of integration of specification and implementation does not separate the *upstream activities* and *downstream activities* [Swartout, Balzer 82], such as in waterfall models that presuppose clear separation of specification from implementation. With the coevolution model, designers concurrently articulate "what" they need to design and "how" to design. The coevolution model distinguishes itself from the *spiral model* [Boehm 88] in its phaseless separation of the activities (see Section 4.4).

Facilitating Communication. An argumentation base and a catalog base of a design environment facilitate long-term indirect communication among designers. Designers communicate through designed artifacts instead of talking or writing directly. Communication is embedded in design artifacts [Reeves 93]. A portion of design rationale articulated and stored by a designer can be used to produce a specification-linking rule, which helps other designers at other times to solve other design problems (see Chapter 6). The catalog base also provides a communication medium among designers. Designers store a design, which helps other designers to produce ideas for a solution. As discussed below, using catalog examples helps designers in performing creative design.

Another kind of communication that the multifaceted architecture can facilitate is through a specification component. A specification component allows designers to specify their problems in terms of the problem domain. For example, KIDSPECIFICATION can be used by end-users, as well

as by designers. In kitchen design, designers use questionnaires written in casual terms in order to communicate with clients.

Software developers and end-users use different *languages*, which causes communication breakdowns between the two stakeholders [Greenbaum, Kyng 91]. Using a prototype system produced by a construction component helps designers and end-users to understand what the other partners are trying to *say*, by having the explicit representation of a partial solution. A specification component of a design environment provides an explicit representation of a problem. Whereas prototypes in the construction component are built by designers, partial problems in the specification component are built by end-users.

Specification-linking rules used in KID facilitate communication between designers and end-users by providing mapping between the construction and the specification. The specification-linking rules represent interdependencies among specification and construction. End-users specify their problems in the specification component and designers build their solutions in the construction component. Specification-linking rules can detect inconsistencies between the two in a form of specific critics identified by RULE-DELIVERER. CASE-DELIVERER also helps end-users to understand the language that designers use by providing a concrete representation of a solution (a catalog example) that is retrieved based on their problem specification.

10.3.1. Reuse-Based Object-Oriented Design Environment

One of the virtues of object-oriented programming resides in its high reusability [Meyer 87]. Complex systems will evolve much more rapidly from simple systems if there are stable intermediate forms [Simon 81]. Object-orientation helps software developers analyze, design, and implement software not by building objects from scratch but by reusing objects from class libraries as "stable intermediate forms." The information overload problem described in Section 3.3, however, has been identified as a serious problem in object reuse.

Empirical evidence shows that software designers cannot identify what abstractions are relevant to the problem and where or to what extent they should be generalized or specialized [Fischer et al. 93a]. For example, a case study reported [Seamail 92] in using the SMALLTALK-80 environment [Goldberg, Robson 83] showed that although the environment provides a large class library for designers, only a small fraction of the library was used. In the study, when subjects designed a complex heating system, they defined a class *''switch''* and defined associated methods, whereas the library had a class called *''binary-state''* that provided exactly the same functionality as the class that they defined. Designers do not know *what* to look for and do not know the existence of relevant objects, and therefore, they are not motivated to look for them.

Object-oriented programming environments must be ''reuse-based;'' reuse of objects from class libraries should not be considered as a secondary activity but rather be a core activity in developing software. In most of existing object-oriented programming environments, much effort goes into

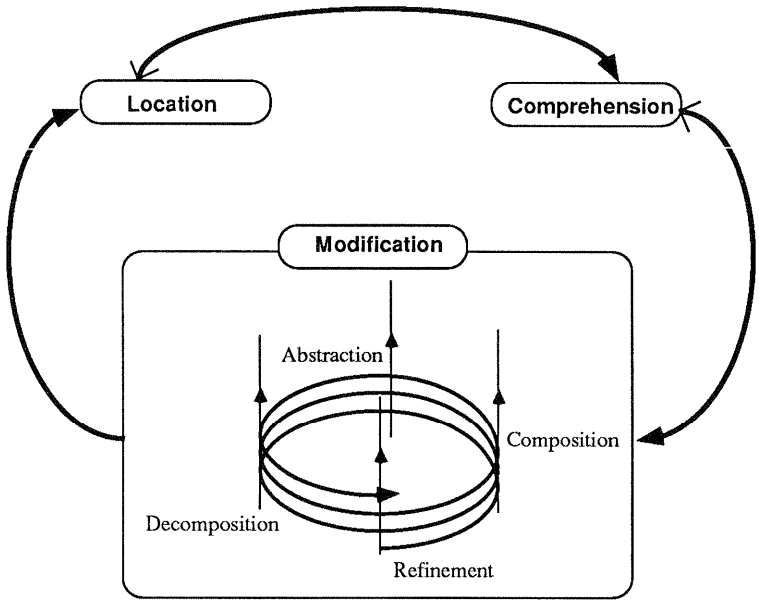


Figure 10-2: The Location-Comprehension-Modification Cycle

supporting the "creation" of objects; they do not support development processes from a "reuse and evolution" point of view. Specifically, two major problems are: (1) *use*: lack of support for reusing objects, such as locating, comprehending, and modifying reusable objects; and (2) *produce*: lack of support for developing objects to be reusable and identifying whether developed objects are abstract and suitable to be stored in a class library.

There are a number of obstacles that make software reuse difficult [Fischer 87b]. In many software design projects, the strict waterfall model cannot be followed because specifications are developed along with and not prior to the other design phases. Thus, a complete specification is not available for retrieving reusable objects and a system for a reuse must deal with partial, preliminary specifications. Another observation is that we cannot obtain reusable software at no cost [Nielsen, Richards 89]. A certain amount of additional work is required if we plan to reuse software, which, on the other hand, must not exceed the cost of creation from scratch.

10.3.2. Support for Use: The Location-Comprehension-Modification Cycle

To support use of reusable objects, the development model in object-oriented programming should be integrated into a software reuse cycle consisting of location, comprehension, and modification [Fischer et al. 92c]. Figure 10-2 illustrates the cycle of these activities. Software developers have to locate reusable objects, comprehend the retrieved objects, and modify them according to their current needs. The modification process includes not simply selecting elements from the palette and positioning them as in the CONSTRUCTION subsystem of KID, but includes processes in which software designers actually evolve objects by modifying existing objects and adding new objects.

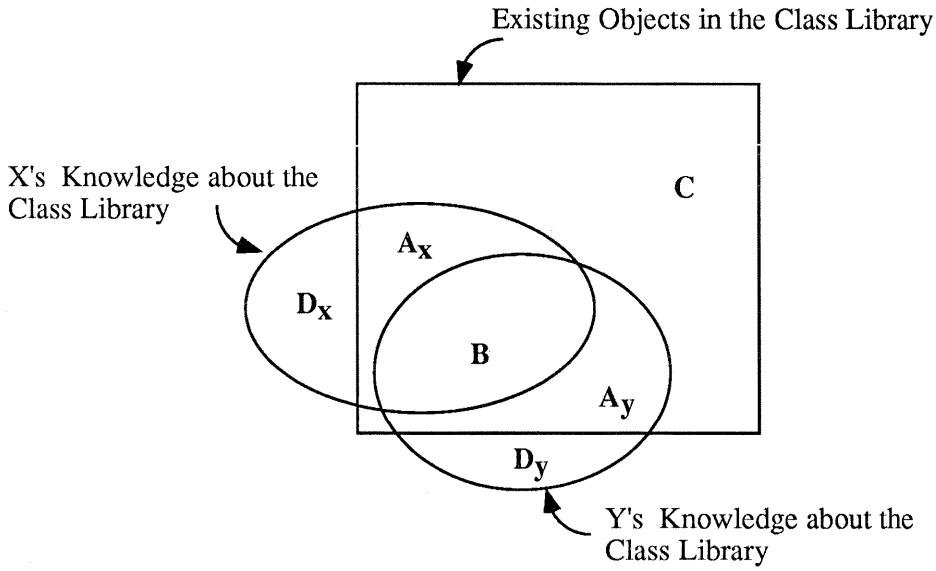


Figure 10-3: Programmer's Perception of Information Space

The cycle of processes must be supported as an integrated whole in an object-oriented programming environment. The shared understanding about a design task provided by the multifaceted architecture increases the quality of support for location, comprehension, and modification of the objects.

Location. Object-oriented design environments are high-functionality systems with a potentially very large number of design objects. As discussed in Section 3.3, the large and growing discrepancy between the amount of potentially relevant information and the amount any one designer can know and remember limits the ability of designers to take advantage of this information.

Figure 10-3 is a variation of Figure 3-1, that illustrates the problem when more than one developer work together using the environment. When more than one developer use an object-oriented design environment, the problem becomes more complex. When one designer (Y) creates an object that is useful to another designer (X), X may not be aware that Y has created the object (the area A_y in Figure 10-3) and may create another object that is very similar to the one Y has created. As discussed in Section 3.3, to support designers in these situations, the object-oriented development environment must provide a combination of *access* and *delivery* mechanisms.

The CASE-DELIVERER mechanism provided by KID addresses this problem. As discussed in Chapter 3, existing access mechanisms such as browsing are not enough to cope with the information overload problem. CASE-DELIVERER supports designers in reusing existing objects from class libraries. A catalog base in KID provides a library of predefined objects. A specification component allows designers to specify their problem. CASE-DELIVERER can then provide designers with objects from a catalog base relevant to the specified problem using the specification-linking rule mechanism. For example, with the above problem, suppose designers use a design

environment for designing heating systems. Using the specification component, designers can specify that they need a *''switch.''* In the argumentation base, there can be an argument saying that if someone needs to implement a switch, a class *''binary-state''* can be used. In the construction component for the design environment, the system can deliver the binary-state class to the designers even if the designers did not search for useful objects.

For effectively supporting the location of reusable objects, more sophisticated *information access* techniques are also necessary. As described in Section 3.3, information access is the designer-initiated location of information when designers perceive a need for information. For example, INFOSCOPE [Henninger 93] helps users to locate EMACS-LISP functions with a query consisting of abstract keywords. The system uses spreading activation to identify relevance between LISP functions to such abstract terms. This dynamic inference of relevance has been found powerful in supporting software programmers [Henninger 93]. Information provided through the specification component of the multifaceted architecture serves as a query that is submitted to this retrieval mechanism.

Comprehension. Most of current object-oriented programming environments provide little support for comprehending retrieved objects. They may provide detailed descriptions about an object itself, such as how the object is implemented and what components the object has, but little information is provided in terms of how the object can be *used.*

Examples of using an object provide useful information for how to use the object. For example, a catalog example in KID provides information about how to use a corner cabinet. Because design units and catalog examples in KID are simple enough to understand, no special support has been provided except the functionality that allows designers to evaluate catalog examples in terms of the current specification (see Section 7.1.3).

When examples are complex, such as an application class object consisting of lines of codes, designers need to analyze the examples more carefully to build an analogy between the examples and the current task. The EXPLAINER system [Redmiles 92] is an approach for supporting designers in understanding software objects by providing various perspectives, for example, the problem domain perspective or the program construct perspective. The simulation component mentioned in Section 10.4.2 can also help designers to understand the located objects.

Argumentation provided by the argumentation base of the multifaceted architecture also provides useful information for understanding retrieved objects. Argumentation provides design rationale [Fischer et al. 91b; Lee 90; Yakemovic, Conklin 90] — a way to associate the designer's reasoning process with the final form of the object. Design rationale can make explicit decisions that influenced the final form of the object, which were otherwise implicit. In addition to explicating the underlying meaning of an object and allowing designers to reuse the object, design rationale associated with the object can also be ''reused'' [Nakakoji, Ostwald 92]. That is, design rationale for a previous design effort can be used to solve a new design problem. Although each design is

unique in principle, designers often face problems similar to what has been faced before. Design rationale can be seen as an explanation of how a designer solved a particular problem in the past.

Thus, objects stored in class libraries of an object-oriented programming environment must provide flexible and powerful representations, including examples and annotations such as design rationale.

Modification. After a design object has been located and understood, software developers have to modify it to accommodate the task at hand. Modification in KID is a process to position design units in the work area and to select answers in the specification component. In software design, however, modification of a partial construction is an iterative process of *specialization*, *composition*, *abstraction*, and *decomposition* of reusable objects [Aoki 93; Aoki, Nakakoji 93]. Any of the four processes may lead to further location of other related objects or require further comprehension of the objects.

Figure 10-2 illustrates the four processes iteratively exercised in a modification process leading to object evolution. The *refinement* process uses inheritance mechanisms to specialize or improve the existing classes to accommodate to the task at hand. After identifying a superclass to be specialized which has similar behaviors (functions) or states (structure) to what software designers want, designers create and define new classes as subclasses to it associated with *is-a* or *is-kind-of* relationships. The *abstraction* process generates an abstract class from existing classes by identifying patterns common to the behaviors or states of the existing classes. The *composition* process creates a complex object by combining existing objects into one object. Objects are associated with *has-a* or *part-of* relationships. Finally, the *decomposition* decomposes the internal states or functions of a large complex object into components to be reused.

Each step is a result of a design decision made. With existing object-oriented programming environments, little or no support is provided for *guiding* these processes. In the multifaceted architecture, the argumentation base allows software designers to store arguments to annotate these processes if needed. Critics, or other types of knowledge, should be delivered as feedback to the software designers about objects that are currently under development and information about how to evolve objects so that they will be reusable.

10.3.3. Support for Production: Metrics for "Reusable" Objects

Metrics proposed by Aoki [Aoki 93; Fischer et al. 93b; Aoki, Nakakoji 93] can be used to identify which objects should be stored in a palette of a construction component as reusable abstract library components or in a catalog base as concrete application examples to support understand how to use those abstract objects.

Two metrics are the *Reference Factor* (RF), which measures the degree of composition and decomposition, and the *Hierarchy Factor* (HF), which measures the degree of refinement and abstraction (see Figure 10-4). Aoki and Nakakoji [1993] has demonstrated that applying the two metrics to classes of the SMALLTALK-80 library and those created for a specific project clearly illustrates that

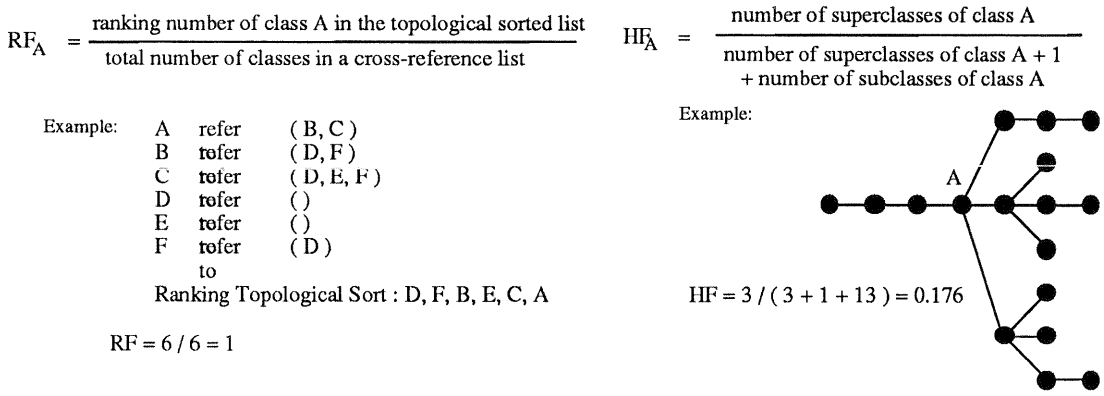


Figure 10-4: Definition of Reference Factor (RF) and Hierarchy Factor (HF)

The *reference factor* (RF) is the metrics for the composition and decomposition of a class, and defined in terms of the total number of classes in the reference paths and the rank order of a class in a partially ordered reference sequence. The value of RF is between 0 and 1; the value becomes larger when the class references more classes, or is referenced by less classes. The *hierarchy factor* (HF) is the metrics for the refinement and abstraction of a class, and is defined in terms of the number of its superclasses and subclasses. The value of HF is between 0 and 1; the larger the value is, the more abstract the class is [Aoki, Nakakoji 93].

---

the former has strongly shown the library-characteristic and the latter has shown the application-characteristic. Thus, the measurement can be embedded into the multifaceted architecture as an analyses mechanism, and those measured values can be used for location and comprehension processes.

10.4. Challenges

In this chapter, I have discussed how the multifaceted architecture can be applied to software design to extend and enhance the object-oriented development paradigm without mentioning a major challenge: *how to develop such a design environment in the first place.*

Several questions should be considered before deciding to build a design environment for software design for a particular domain. A prerequisite question to be answered is whether building software for the domain takes place often enough; otherwise, it is not worth building the design environment. If the cost outweighs the benefit, one should not build a design environment; then, *the multifaceted architecture is useless for such domains.* If one decides the benefit outweighs the cost, then, the subsequent questions are: *how does one develop such a design environment?* What is an appropriate representation for a specification, construction, and a catalog base? What would critics look like in this domain? Is computational power enough to process a large number of software objects?

In this section, I first discuss future software development by introducing such a design environment. Second, I discuss necessary refinement of the architecture when applied to the software design, including the need for a simulation component and the need for a process-guiding capability. Next, I briefly discuss types of domains and problems and challenges in building the

domain-oriented design environment. Finally, I conclude that the environment can serve not only as a starting point, but also as a check point to see that necessary information has been constructed throughout the software development project.

10.4.1. Future Software Development

Figure 10-5 illustrates how I view the role distributions of future software development. Introducing a level of domain-oriented design environment and a level of component tools for the environment specializes skills of software developers. There would be a separation of software developers into those who (1) build component tools, (2) build a domain-oriented design environment using those component tools, and (3) build an application using the domain-oriented design environments.

Design takes place between each level. Tool builders *design* environment component tools based on requirements articulated by environment builders. The environment builders *design* domain-oriented design environments in accordance with requirements articulated by application builders. The application builders in turn design application software based on end-users' requirement specifications. As discussed throughout this dissertation, requirement specification and solution construction must be integrated; therefore, those stakeholders must work together to produce quality artifacts.

Although it is vital to facilitate communication among people in the two adjacent levels, as shown in Figure 10-5, the goal is to reduce the amount of communication required between people in nonadjacent levels. People in each level use different *languages* to achieve their design. The communication distance represented by the number of levels involved represents the number of languages with which the participants have to deal. The more languages, the larger the risk that miscommunication may take place. We have to reduce the problem of miscommunication by reducing the communication distance.

The multifaceted architecture provides a specification component and construction component together with specification-linking rules to facilitate the communication between the two levels of environment builders and application builders. A specification component of a design environment facilitates shared understanding not only between designers and the design environment, but also between designers in the adjacent levels.

10.4.2. Refinement of the Architecture

The multifaceted architecture should provide a *simulation* component for domains in which enactment of the design is possible. The simulation component allows designers to carry out "what-if" games by simulating usage scenarios with the artifact being designed. Feedback from the simulation component complements the results of *seeing* using the design knowledge (argumentation and catalog examples). Design knowledge provided in the argumentation base represents only "articulated" knowledge. Design knowledge represented in the catalog base provides only what some-

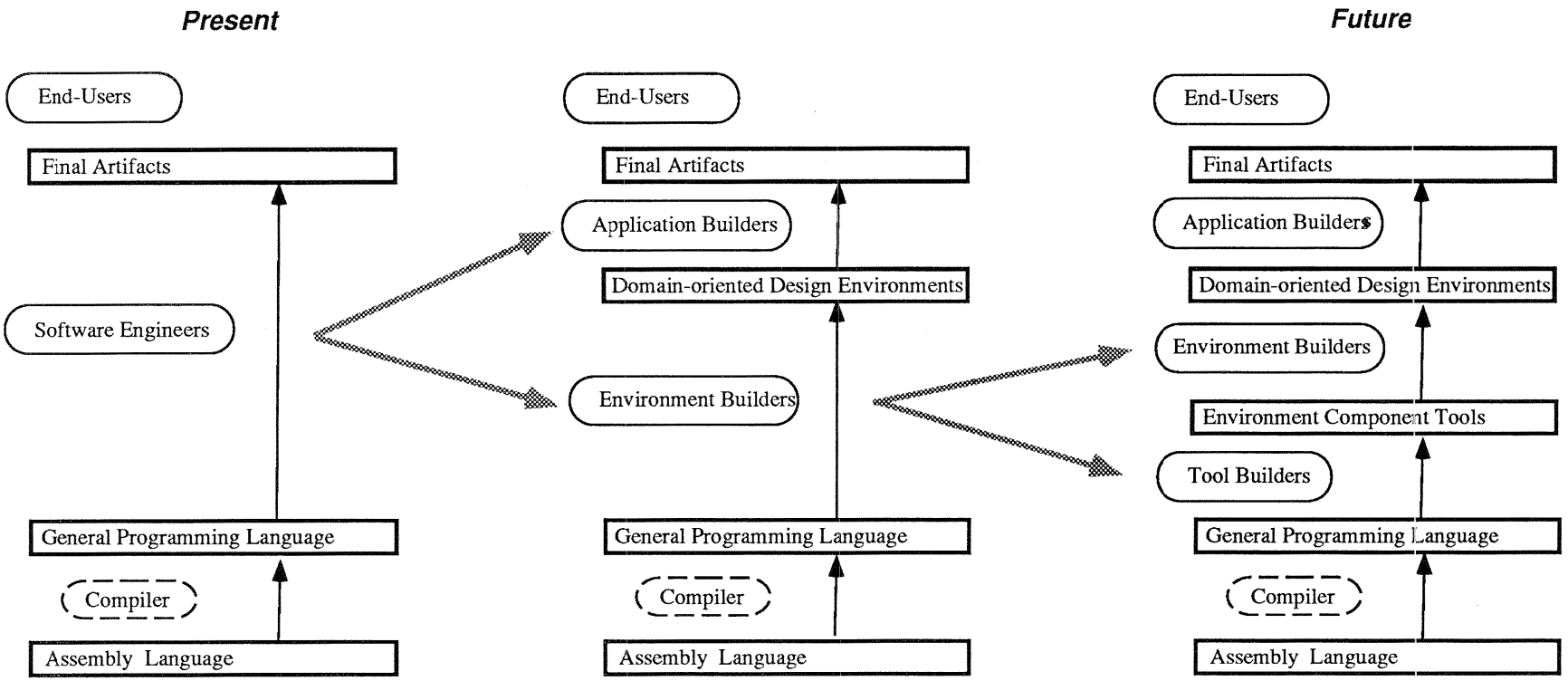


Figure 10-5: Specialization of Software Engineers

body else has done. The simulation component can uncover unforeseeable situations and help designers to understand *function-as-involvement* [McLaughlin 92], which is described in Chapter 2.

A specification component of a design environment can increase the quality of the simulation. Simulating the usage of designed artifacts or executing the designed artifact must take into account what the problem is. Designers have to be able to *see* the results of the simulation in terms of their problem specification, and the design environment can help the designers in doing so by using the information provided in the specification component.

As observed in the user study (see Chapter 8), KID is quite complex, and subjects get lost while using the system. Delivery of information is a type of process guidance because during a design, designers can trace relationships among delivered information and naturally attend various subsystems of KID. Other design environments, such as FRAMER [Lemke 89], have explored the notion of a checklist that explicitly guides users as to which system-component to attend next. Process guidance techniques that have been studied in software engineering [Osterweil 87; Heineman et al. 91] should be applied to the multifaceted architecture to provide process guidance.

10.4.3. Types of Domain and Evolution of Design Environments

In order to build a domain-oriented design environment, we need to identify and analyze the domain in consideration along two dimensions:

Stability. A *stable* domain is a well-established domain such as kitchen design. Its domain abstractions are well identified and the design principles and standards are well known. The design knowledge is taught through textbooks or design seminars. An *unstable* domain, in contrast, is a domain that is relatively newly established, is still under exploration, or heavily depends on state-of-the-art technologies. An example of this type of domain is user-interface design. Although some domain knowledge and design experts exist for designing user-interfaces, there has not been a stable standard or globally acknowledged rules yet. Domain abstractions and components are constantly changing.

Maturity. A *mature* domain has well-established standards and representations for design activities and solution forms, but not necessarily design knowledge such as heuristics leading to solutions. For example, although the computer network design domain is unstable, it is a *mature* domain because the meaning of designing a computer network is clearly well-understood. Designers need to produce logical and physical maps of a computer network using nodes and links regarding computer hardware and cables. An example of an *immature* domain is a software system design for a large corporate organization. Designers do not know which kinds of operations will be needed, or how they are organized; designers do not know how to represent such information to start with. Designers have neither the design knowledge (principles and abstractions), nor ideas of how a solution form would look.

It is difficult to define the boundary for a domain when building a design environment. The defini-

tion of a domain that a community adopts depends not only on technical factors, but also on social and economic factors [Prieto-Diaz, Arango 91]. Even with a mature, stable domain such as kitchen design, there are many types of kitchen design involved, such as residential kitchen design, commercial kitchen design, customized kitchen design, and kitchen design for ready-built houses. There may be as many domains as kinds of professions exist, and as professions have only vague boundaries, so do domains.

As discussed in Chapter 2, a design domain can never be complete and stable, even with a mature, relatively stable domain. For example, in kitchen design, design knowledge is first synthesized in a booklet *''Functional Kitchen Storage''* by Heiner and McGullough, published in 1948. After using the identified thesis for more than four decades, the National Kitchen and Bath Association (NKBA) has published a new booklet including ''new rules'' in order to accommodate changes that have emerged as contemporary kitchen technologies integrate with family roles, preferences, and management styles. New rules include a kitchen plan for two, proper location of microwave ovens, and the effectiveness of downdraft ventilation systems versus overhead ventilation systems [NKBA 92]. For a domain-oriented design environment, supporting evolution of its knowledge base is critical in order to accord with the changes of the domain itself.

Figure 10-6 illustrates how the evolution of a design environment should be supported.

Seeding: Initializing a Design Environment. Research has been done to explore how to define a domain and its boundary on a computer system by identifying important concepts, abstractions, and semantics in the domain and storing them into the system. One approach is to use *knowledge-engineering techniques*, in which *knowledge engineers* analyze the real world practice and identify relevant design knowledge, in collaboration with *domain experts*. The EVA (Evolving Artifact) approach [Ostwald, Burns, Morch 92], for example, has been proposed to challenge this issue by supporting *mutual education* between knowledge engineers and domain experts. Another approach is to use domain analysis techniques [Prieto-Diaz, Arango 91] that stem from the software reuse community. The goal is to identify the reusable portions of knowledge to save tasks for transforming low-level computer primitives to abstract data types.

In some sense, the domain analysis techniques and the knowledge-engineering techniques are the two extremes. The domain analyses approach tries to identify domain abstraction from software objects that are built on a computer system by software developers, leading to theory formation. The knowledge-engineering approach, on the other hand, tries to articulate knowledge that exists in an expert designer's head and in the world in order to build it onto a design environment. Without articulated knowledge, they cannot formulate theories. Without theories, articulated design knowledge cannot be effectively used. These two approaches, therefore, are necessary in order to complement each other.

Reseeding: Accommodating the Change of the Domain. Through practicing design using a design environment, designers (users of the design environment) gain an understanding of the

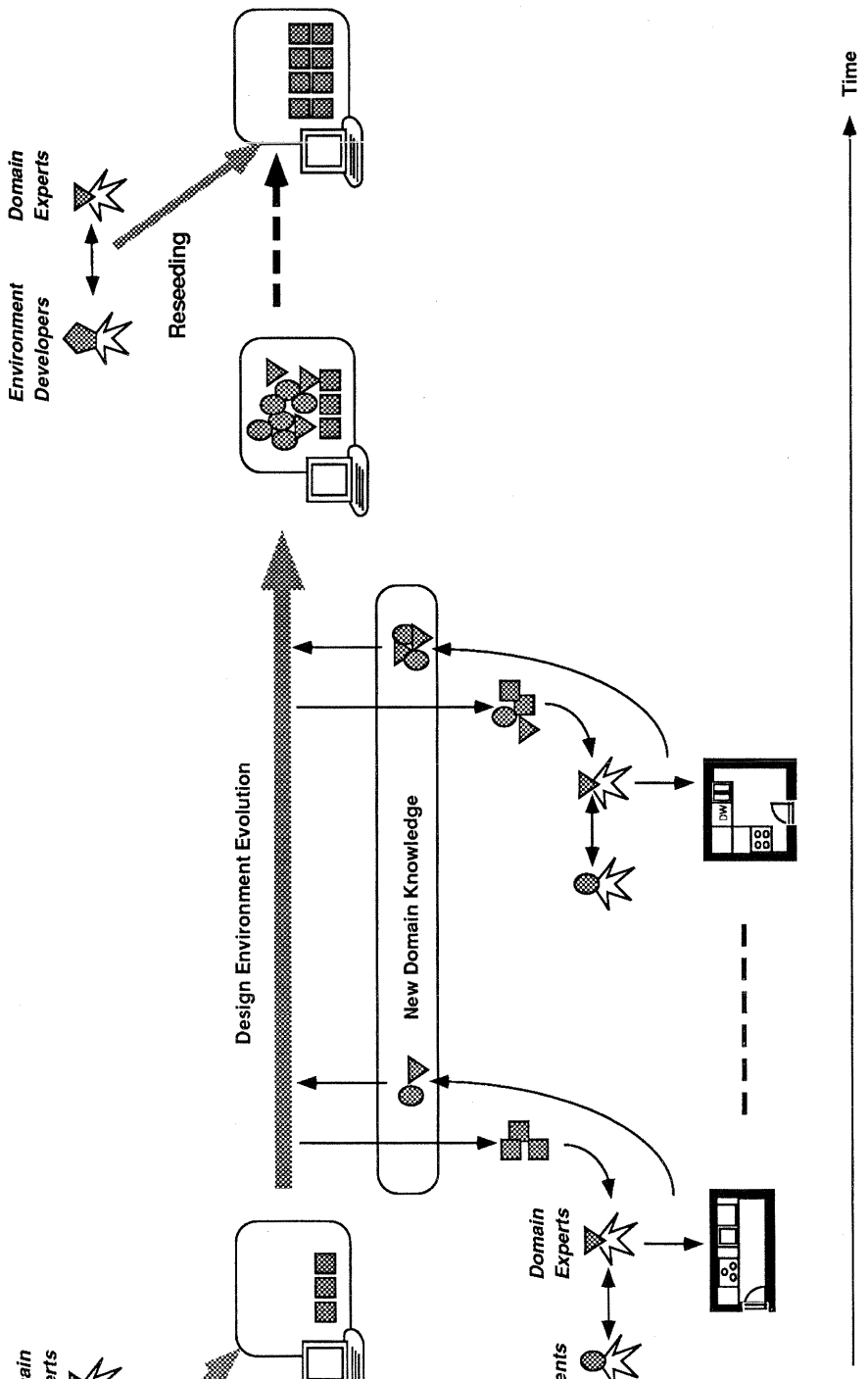


Figure 10-6: Evolution of a Knowledge Base in a Design Environment

Through practicing design with a design environment, designers accumulate design knowledge in knowledge bases. Refinement of the knowledge base (*reseeding*) is occasionally done by designers and knowledge engineers.

domain. Design practice changes over time as society grows. Designers may realize the lack of a fundamental functionality of the design environment, may introduce inconsistent design knowledge, or may become aware of obsolete design principles. The boundary of what a designer does and what the system does in a semiformal architecture will change over time because our understanding of what can be formalized grows over time as designers reflect on their jobs and establish routines.

Evolution through the Use of the Design Environment. As designers keep using a design environment, design experience can be accumulated in the system. A typical example of this type of knowledge includes design rationale [AAAI Workshop 92] and design artifacts as cases. For example, in KID, design rationale that emerges during specific design tasks can be accumulated in the argumentation base using KIDSPECIFICATION, and designed artifacts can be stored in the catalog base. Designers usually know more than they can say [Polanyi 66]. As observed in the study presented in Chapter 8, this tacit knowledge is triggered by new design situations and by breakdowns that occur as they engage in a design process. Thus, while design environments are being used, their domain knowledge should be increased and refined by interacting with designers. Designers must be able to add and modify their domain knowledge, not by "programming," nor with the help of knowledge engineers, but in response to breakdowns.

The KID approach is directly applicable to a relatively mature, stable domain because the multifaceted architecture presupposes that domain abstractions have already been identified to be used as design primitives in the construction component. On the other hand, even with immature or unstable domains, the multifaceted architecture can be used as a guideline. Instead of prebuilding a design environment for such a domain, software designers develop a design, and the outcome of the design should provide representations covered by the multifaceted architecture. In other words, through the design process, designers can try to identify appropriate representations for a specification and a construction, arguments, and cases for later reuse.

10.5. Summary

In this chapter, I discuss the applicability and extensibility of the multifaceted architecture that I have been exploring in the context of KID, a kitchen floor plan design environment. Despite the simplicity of the domain, KID was powerful enough to reveal many research issues and challenges that are generally applicable to design, including software design. The architecture itself is domain-independent for instantiating a domain-dependent design environment. The idea of having a specification component, a construction component, a catalog base, an argumentation base, and knowledge (or information) delivery mechanisms that integrate them can be applied to a wide variety of domain-oriented design environments. I claim specifically that the approach can form a synergistic whole if built on top of the object-oriented computational substrate.

In this dissertation, I have explained how the multifaceted architecture improves current design practice to produce *better design*, using KID as a prototype. Another view of the approach is to

practice design while having the multifaceted architecture as an objective and to accumulate design information; this approach will lead to *better design practice* in the future. With software design in which a domain boundary is not clear, the latter usage of the architecture seems more promising. This research provides a starting point to go beyond object orientation for the next generation of software engineering.

Chapter 11

Conclusion

In conclusion, two points are worth mentioning: creative design facilitated by knowledge delivery, and potential users of KID.

Creative design facilitated by knowledge delivery. The KID approach amplifies designers' creativity by providing designers with information relevant to their task at hand. *Reminding* is considered to be one of the most crucial aspects supporting creativity [McLaughlin, Gero 89; Boden 91]. Reminding is supported in our environments by (1) breakdowns (signaling the violation of a design rule), (2) accessing domain concepts in the argumentation, and (3) locating interesting examples in the catalog. Analyzing this information, designers can draw analogies and/or discover new ideas leading them to create new moves in their design activities that they might have overlooked otherwise.

Creative design should be both *innovative* and *valuable*. By exploiting the information contained in a partial specification and a partial construction, KID can provided information that has the potential to be simultaneously valuable and innovative. There is a spectrum of how accurately and closely related the presented information should be to the designers' task at hand. At one end of the spectrum is information retrieved based on precise queries formed by designers. At the other end of the spectrum is arbitrary information presented to the designers [Owen 86]. If designers are given arbitrary information, they may likely get innovative ideas from the information, but the ideas may not be valuable. If designers are given information based on their precise queries, they may get valuable but less innovative ideas from the information.

Delivering information relevant to the task at hand thus provides designers with *cybernetic* serendipity [Edmonds 93], which helps designers in searching for a new solution by projecting new meanings into the information they might have known before. An integration of delivery and access mechanisms of KID, consisting of RULE-DELIVERER, CASE-DELIVERER, and CATALOGEXPLORER reminds designers of different types of information, such as catalog examples and argumentation.

Potential users of KID. Through development of KID, two potential usage situations have been identified:

- *Training novice designers.* KID has originally been aimed at use by designers. In the user study, however, the expert designer mentioned that KID provides a good learning substrate for a new designer. I define novice designers as professional designers who are new to the domain but are willing to learn domain knowledge, and distinguish them from naive designers, who are neither experts nor interested in becoming experts. Because knowledge delivered by KID is assumed to be relevant to the task at hand, it is easier for designers to contextualize the delivered knowledge and understand it. Thus, the interaction styles that KID provides support learning-in-demand.

- *Facilitating communication between end-users and designers.* As discussed in Section 10.3, because KID has specification and construction components, KID facilitates communication between end-users and designers. Suppose both a client and a designer sit together in front of KID and use the system. The client fills out the specification component, and the designer starts constructing a floor plan in the specification component. Then KID fires a critic saying there is a conflict between the partial specification and construction. This delivered knowledge invokes communication between the two participants, and because they have the concrete objects to work on, that is, a concrete representation of a problem specification and solution construction, their communication goes smoothly with the help of knowledge delivery mechanisms.

Schoen [1992] has discussed what and how computers can support design. Schoen listed four essential elements required for building a *''promising''* design assistant computer system:

1. to produce environments that enhance the designer's seeing-drawing-seeing,
2. to create ''microworlds'' that can be programmed to function as design worlds, extending the designer's ability to construct and explore them,
3. to provide a system that extends the designers' repertoire of prototypes and enhances designers' ability to explore and bring them into transaction with particular design situations, and
4. to create an environment that helps the designers to discover and reflect upon their own design knowledge.

KID addresses all four elements. First, KIDSPECIFICATION and CONSTRUCTION allow designers not only to *draw* but to *frame* explicit representations of both their problem specifications and solution constructions. Having an explicit representation of the problem specification lessens designers' cognitive load of remembering and manipulating the problem, and thus helps designers to see their partially framed design. Second, KIDSPECIFICATION in conjunction with CONSTRUCTION embodies knowledge representations of the task at hand, and thus provides microworlds in which designers can be engaged. Design objects provided by KIDSPECIFICATION and CONSTRUCTION are extensible by designers; thereby KID allows designers to adjust the microworlds to their own way. Third, the argumentation base and catalog base of KID provide a repository of design knowledge that has been accumulated through previous design efforts. The RULE-DELIVERER and CASE-DELIVERER mechanisms of KID support designers to explore and use the stored design knowledge by delivering the portions of knowledge relevant to designers' task at hand. Finally, the delivery mechanisms provide useful design knowledge, which triggers designers to discover breakdowns in their partial design, verify them using the delivered knowledge, and reframe their design when necessary.

In this research, I explored the role of a specification component in KID. The above characterizations of KID have been enabled by having the specification component in its architecture. KIDSPECIFICATION is designed as a hypermedia interface for the argumentation base, which provides *semi-formal* structure for a design space. Specification-linking rules are used to integrate the specification component with the other components of KID, and knowledge delivery

mechanisms are realized using the specification-linking rules. The rules are automatically derived from the content of the argumentation base by using *serve relationships* that the semi-structured information space inherently provides.

The user study showed that KIDSPECIFICATION helped designers to understand the problem better early in the design process. Reading and answering questions given by KIDSPECIFICATION prevented the subjects from overlooking important considerations, and yet did not hinder them from seeing the problem in new ways. The information provided in KIDSPECIFICATION, in turn, increased the quality of knowledge delivery by giving the system more information about the subjects' task at hand. In addition to the delivered knowledge itself, the reason for its presentation triggered designers to reflect on their problem. Delivering knowledge also supported the designers to become aware of tacit design knowledge. They often reacted to delivered knowledge by arguing against it in terms of their task at hand. When given an object to think with, people start thinking about it and trace associations, which may be linked to tacit part of design knowledge. Thus, it was easier for the designers to become aware of tacit design knowledge and to articulate it, than to start articulating design knowledge given no context. Finally, it was found that the success of knowledge delivery depends on its integration with good access mechanisms. Designers were often motivated to explore related knowledge spaces in KID after having the delivered knowledge. When the system provided services for the designers with no additional effort, the designers appreciated it, seemingly regardless of the quality of the delivered information. When they had a specific task in their mind, however, they did not feel comfortable if the system's capability limited what they could do.

The KID design environment has been designed to support the seeing-framing-seeing cycle of design processes based on the human-computer cooperative problem-solving paradigm. The specification component in conjunction with the construction component increases the amount of shared understanding of the task at hand between designers and the design environment by providing explicit representations of a specification and construction. The shared understanding enables the knowledge delivery mechanisms to deliver the design knowledge relevant to the task at hand. The delivered design knowledge helps designers to reflect on their partial design, and thus supports designers in coevolving problem specification and solution construction.

The KID design environment has been developed as the multifaceted architecture has been evolved. KID is integrated by establishing many links among individual subsystems. Its knowledge representations are distributed among the components because of the historical background. The presentation of KID in this dissertation is not intended to show that KID is *the* answer to implementing the idea of the architecture; KID is *a* prototype system, which demonstrates that the idea *can be* implemented on a computational substrate. By having a working prototype at hand, we can *see* what the real problems are, and we can gain an understanding of the problems of design.

References

[AAAI Workshop 92]
Working Notes of the AAAI 1992 Workshop on Design Rationale Capture and Use, AAAI, San Jose, CA, July 1992.

[Akscyn, McCracken, Yoder 88]
R.M. Akscyn, D.L. McCracken, E.A. Yoder, *KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations*, Communications of the ACM, Vol. 31, No. 7, July 1988, pp. 820-835.

[Alexander 64]
C. Alexander, *The Synthesis of Form*, Harvard University Press, 1964.

[Aoki 93]
A. Aoki, *Introduction to Object-Oriented Analyses and Design*, Sofuto-Research Center, Inc., Tokyo, Japan, 1993, (in Japanese).

[Aoki, Nakakoji 93]
A. Aoki, K. Nakakoji, *Empirical Study on Object Evolution Using Metrics for Reuse-Based Object-Oriented Design Environment*, Proceedings of Symposium on the Foundations of Software Engineering, Los Angeles, CA., December 1993, (Submitted).

[Arias 93a]
E.G. Arias (ed.), *The Meaning and Use of Housing*, Avebury Ashgate Publishing Limited, England, 1993.

[Arias 93b]
E.G. Arias, *User Group Preferences and Their Intensity: The Impacts of Residential Design*, in E.G. Arias (ed.), *The Meaning and Use of Housing*, Avebury Ashgate Publishing Limited, England, 1993, ch. 7.

[Belady 85]
L. Belady, *MCC: Planning the Revolution in Software*, IEEE Software, November 1985, pp. 68-73.

[Boden 91]
M. Boden, *The Creative Mind: Myths & Mechanisms*, Basic Books, 1991.

[Boehm 88]
B.W. Boehm, *A Spiral Model of Software Development and Enhancement*, IEEE Computer, Vol. 21, No. 5, May 1988, pp. 61-72.

[Bonnardel 91]
N. Bonnardel, *Criteria Used for Evaluation of Design Solutions*, Designing for Everyone and Everybody: 11th Congress of the International Ergonomics Asssociation, Y. Queinnec, F. Daniellou (ed.), Paris, France, 1991.

[Bonnardel 93]
N. Bonnardel, *Knowledge Elicitation Through Project Transfer: An Experimental Study*, The International Review: Behavior and Information Technology, 1993, (forthcoming).

[Booch 91]
G. Booch, *Object Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1991.

[Carroll, McKendree 87]
J.M. Carroll, J. McKendree, *Interface Design Issues for Advice-Giving Expert Systems*, Communications of the ACM, Vol. 30, No. 1, January 1987, pp. 14-31.

[Conklin, Begeman 88]
J. Conklin, M. Begeman, *gIBIS: A Hypertext Tool for Exploratory Policy Discussion*, Transactions of Office Information Systems, Vol. 6, No. 4, October 1988, pp. 303-331.

[Cox 86]
B.J. Cox, *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley Publishing Company, Reading, MA, 1986.

[Cross 84]
N. Cross, *Developments in Design Methodology*, John Wiley & Sons, New York, 1984.

[CSTB 88]
Computer Science and Technology Board, *The National Challenge in Computer Science and Technology*, National Academy Press, Wathington, DC, 1988.

[CSTB 90]
Computer Science and Technology Board, *Scaling Up: A Research Agenda for Software Engineering*, Communications of the ACM, Vol. 33, No. 3, March 1990, pp. 281-293.

[Curtis, Krasner, Iscoe 88]
B. Curtis, H. Krasner, N. Iscoe, *A Field Study of the Software Design Process for Large Systems*, Communications of the ACM, Vol. 31, No. 11, November 1988, pp. 1268-1287.

[Devanbu et al. 91]
P. Devanbu, R.J. Brachman, P.G. Sefridge, B.W. Ballard, *LaSSIE: A Knowledge-Based Software Information System*, Communications of the ACM, Vol. 34, No. 5, 1991, pp. 34-49.

[Domeshek, Kolodner 92]
E.A. Domeshek, J.L. Kolodner, *A Case-Based Design Aid for Architecture*, in J. Gero (ed.), *Artificial Intelligence in Design'92*, Kluwer Academic Publishers, Netherlands, 1992, pp. 497-561.

[Draper 84]
S.W. Draper, *The Nature of Expertise in UNIX*, Proceedings of INTERACT'84, IFIP Conference on Human-Computer Interaction, Elsevier Science Publishers, Amsterdam, September 1984, pp. 182-186.

[Draper, Norman 84]
S.W. Draper, D.A. Norman, *Software Engineering for User Interfaces*, Proceedings of the 7th International Conference on Software Engineering (Orlando, FL), IEEE Computer Society, Los Angeles, CA, March 1984, pp. 214-220.

[Edmonds 93]
E. Edmonds, *Cybernetic Serendipity Revisited*, in T. Dartnall (ed.), *Artificial Intelligence and Creativity*, Kluwer Academic Publishers, Netherlands, 1993, (in press).

[Ericsson, Simon 84]
K.A. Ericsson, H.A. Simon, *Protocol Analysis: Verbal Reports as Data*, The MIT Press, Cambridge, MA, 1984.

[Fickas, Nagarajan 88]
S. Fickas, P. Nagarajan, *Critiquing Software Specifications*, IEEE Software, Vol. 5, No. 6, November 1988, pp. 37-47.

[Fischer 87a]
G. Fischer, *An Object-Oriented Construction and Tool Kit for Human-Computer Communication*, ACM Computer Graphics, Vol. 21, No. 2, April 1987, pp. 105-109.

[Fischer 87b]
G. Fischer, *Cognitive View of Reuse and Redesign*, IEEE Software, Special Issue on Reusability, Vol. 4, No. 4, July 1987, pp. 60-72.

[Fischer 92]
G. Fischer, *Shared Knowledge in Cooperative Problem-Solving Systems - Integrating Adaptive and Adaptable Systems*, Proceedings of 3rd International Workshop on User Modeling (UM'92), E. Andre et al. (eds.), The German Research Center for Artificial Intelligence, Dagstuhl, Germany, August 1992, pp. 148-161.

[Fischer et al. 91a]
G. Fischer, A.C. Lemke, T. Mastaglio, A. Morch, *The Role of Critiquing in Cooperative Problem Solving*, ACM Transactions on Information Systems, Vol. 9, No. 2, 1991, pp. 123-151.

[Fischer et al. 91b]
G. Fischer, A.C. Lemke, R. McCall, A. Morch, *Making Argumentation Serve Design*, Human Computer Interaction, Vol. 6, No. 3-4, 1991, pp. 393-419.

[Fischer et al. 92a]
G. Fischer, K. Nakakoji, J. Ostwald, G. Stahl, T. Sumner, *Embedding Critics in Integrated Design Environments*, 1992, (to appear in The Knowledge Engineering Review Journal).

[Fischer et al. 92b]
G. Fischer, J. Grudin, A.C. Lemke, R. McCall, J. Ostwald, B.N. Reeves, F. Shipman, *Supporting Indirect, Collaborative Design with Integrated Knowledge-Based Design Environments*, Human Computer Interaction, Special Issue on Computer Supported Cooperative Work, Vol. 7, No. 3, 1992, pp. 281-314.

[Fischer et al. 92c]
G. Fischer, A. Girgensohn, K. Nakakoji, D. Redmiles, *Supporting Software Designers with Integrated, Domain-Oriented Design Environments*, IEEE Transactions on Software Engineering, Special Issue on Knowledge Representation and Reasoning in Software Engineering, Vol. 18, No. 6, 1992, pp. 511-522.

[Fischer et al. 93a]
G. Fischer, K. Nakakoji, J. Ostwald, G. Stahl, T. Sumner, *Embedding Computer-Based Critics in the Contexts of Design*, Human Factors in Computing Systems, INTERCHI'93 Conference Proceedings, ACM, 1993, pp. 157-164.

[Fischer et al. 93b]
G. Fischer, D. Redmiles, L. Williams, G. Puhr, A. Aoki, K. Nakakoji, *Beyond Object-Oriented Development: Where Current Object-Oriented Approaches Fall Short*, 1993, (Submitted to the Special Issue of Human-Computer Interaction on Empirical Studies of Object-Oriented Design).

[Fischer, Henninger, Nakakoji 92]
G. Fischer, S. Henninger, K. Nakakoji, *DART: Integrating Information Delivery and Access Mechanisms*, 1992, Unpublished Manuscript.

[Fischer, Henninger, Redmiles 91a]
G. Fischer, S.R. Henninger, D.F. Redmiles, *Intertwining Query Construction and Relevance Evaluation*, Human Factors in Computing Systems, CHI'91 Conference Proceedings (New Orleans, LA), ACM, New York, 1991, pp. 55-62.

[Fischer, Henninger, Redmiles 91b]
G. Fischer, S.R. Henninger, D.F. Redmiles, *Cognitive Tools for Locating and Comprehending Software Objects for Reuse*, Thirteenth International Conference on Software Engineering (Austin, TX), IEEE Computer Society Press, ACM, IEEE, Los Alamitos, CA, 1991, pp. 318-328.

[Fischer, Lemke 88]
G. Fischer, A.C. Lemke, *Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication*, Human-Computer Interaction, Vol. 3, No. 3, 1988, pp. 179-222.

[Fischer, Lemke, Schwab 85]
G. Fischer, A.C. Lemke, T. Schwab, *Knowledge-Based Help Systems*, Human Factors in Computing Systems, CHI'85 Conference Proceedings (San Francisco, CA), ACM, New York, April 1985, pp. 161-167.

[Fischer, McCall, Morch 89]
G. Fischer, R. McCall, A. Morch, *Design Environments for Constructive and Argumentative Design*, Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX), ACM, New York, May 1989, pp. 269-275.

[Fischer, Morch 88]
G. Fischer, A. Morch, *CRACK: A Critiquing Approach to Cooperative Kitchen Design*, Proceedings of the International Conference on Intelligent Tutoring Systems (Montreal, Canada), ACM, New York, June 1988, pp. 176-185.

[Fischer, Nakakoji 91]
G. Fischer, K. Nakakoji, *Empowering Designers with Integrated Design Environments*, in J. Gero (ed.), *Artificial Intelligence in Design'91*, Butterworth-Heinemann Ltd, Oxford, England, 1991, pp. 191-209.

[Fischer, Nakakoji 92]
G. Fischer, K. Nakakoji, *Beyond the Macho Approach of Artificial Intelligence: Empower Human Designers - Do Not Replace Them*, Knowledge-Based Systems Journal, Vol. 5, No. 1, 1992, pp. 15-30.

[Fischer, Nieper-Lemke 89]
G. Fischer, H. Nieper-Lemke, *HELGON: Extending the Retrieval by Reformulation Paradigm*, Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX), ACM, New York, May 1989, pp. 357-362.

[Fischer, Reeves 92]
G. Fischer, B.N. Reeves, *Beyond Intelligent Interfaces: Exploring, Analyzing and Creating Success Models of Cooperative Problem Solving*, Applied Intelligence, Special Issue Intelligent Interfaces, Vol. 1, 1992, pp. 311-332.

[Fischer, Stevens 91]
G. Fischer, C. Stevens, *Information Access in Complex, Poorly Structured Information Spaces*, Human Factors in Computing Systems, CHI'91 Conference Proceedings (New Orleans, LA), ACM, New York, 1991, pp. 63-70.

[Fish 89]
S. Fish, *Doing What Comes Naturally: Change, Rhetoric, and the Practice of Theory in Literary and Legal Studies*, Duke University Press, Durham, NC, 1989.

[Frakes, Gandel 90]
W.B. Frakes, P.B. Gandel, *Representing Reusable Software*, Information and Software Technology, Vol. 32, No. 10, December 1990, pp. 653-664.

[Gero 90]
J.S. Gero, *A Locus for Knowledge-Based Systems in CAAD Education*, in M. McCullough et al. (eds.), *The Electronic Design Studio*, The MIT Press, Cambridge, MA, 1990, pp. 49-60.

[Girgensohn 92]
Andreas Girgensohn, *Modifier: Making Systems End-User Modifiable*, Human Factors in Computing Systems, CHI'92 Conference, ACM, 1992, Submitted.

[Goldberg, Robson 83]
A. Goldberg, D. Robson, *Smalltalk-80, The Language and its Implementation*, Addison-Wesley Publishing Company, Reading, MA, 1983.

[Greenbaum, Kyng 91]
J. Greenbaum, M. Kyng (eds.), *Design at Work: Cooperative Design of Computer Systems*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

[Greeno 89]
J.G. Greeno, *Situations, Mental Models, and Generative Knowledge*, in D. Klahr, K. Kotovsky (eds.), *Complex Information Processing: The Impact of Herbert Simon*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1989, pp. 285-318, ch. 11.

[Greif 88]
I. Greif (ed.), *Computer-Supported Cooperative Work: A Book of Readings*, Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[Gross 90]
M.D. Gross, *Relational Modeling: A Basis for Computer-Assisted Design*, in M. McCullough et al. (eds.), *The Electronic Design Studio*, The MIT Press, Cambridge, MA, 1990, pp. 123-136, ch. 8.

[Grudin 88]
J. Grudin, *Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces*, Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88), ACM, New York, September 1988, pp. 85-93.

[Habraken 87]
N.J. Habraken, *Architecture and Agreement - A Report on Research for New Design Methods (Open-System no Riron to Shuhou)*, Kenchiku Bnka, April 1987, pp. 119-130, (in Japanese).

[Halasz 88]
F.G. Halasz, *Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems*, Communications of the ACM, Vol. 31, No. 7, July 1988, pp. 836-852.

[Hammond 90]
K.J. Hammond, *Case-Based Planning: A Framework for Planning from Experience*, Cognitive Science, Vol. 14, 1990, pp. 385-443.

[Heineman et al. 91]
G.T. Heineman, G.E. Kaiser, N.S. Barghouti, I.Z. Ben-Shaul, *Rule Chaining in MARVEL: Dynamic Binding of Parameters*, 6th Annual Knowledge-Based Software Engineering Conference Proceedings (Syracuse, NY), Rome Laboratory, New York, September 1991, pp. 276-287.

[Henninger 93]
S. R. Henninger, *Locating Relevant Examples for Example-Based Software Design*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, May 1993.

[Hinrichs 90]
T.R. Hinrichs, *Some Criteria for Evaluating Designs*, Proceedings of the Twelfth Annual Conference of the Cognitive Science Society, Boston, MA, July 1990.

[Jakiela 88]
M. J. Jakiela, *Intelligent Suggestive CAD Systems*, Unpublished Ph.D. Dissertation, University of Michigan, 1988.

[Kalay 91]
Y.E. Kalay, *Computational Modalities of Evaluation and Prediction in Design*, CAAD Futures '91 Proceedings, Department of Architecture, Swiss Federal Institute of Technology, Zurich, Switzerland, 1991.

[Kass, Stadnyk 92]
R. Kass, I. Stadnyk, *Using User Models to Improve Organizational Communication*, February 1992. Reviewed paper for User Modeling Workshop.

[Kolodner 90]
J.L. Kolodner, *What is Case-Based Reasoning?*, 1990, In AAAI'90 Tutorial on Case-Based Reasoning, pp. 1-32.

[Kolodner 91]
J.L. Kolodner, *Improving Human Decision Making Through Case-Based Decision Aiding*, AI Magazine, Vol. 12, No. 2, Summer 1991, pp. 52-68.

[Kunz, Rittel 70]
W. Kunz, H.W.J. Rittel, *Issues as Elements of Information Systems*, Working Paper 131, Center for Planning and Development Research, University of California, Berkeley, CA, 1970.

[Lai et al. 88]
K.-Y. Lai, T.W. Malone, K.-C. Yu, *Object Lens: a "Spreadsheet" for Cooperative Work*, ACM Transactions on Office Information Systems, Vol. 6, No. 4, October 1988, pp. 332-353.

[Lee 90]
J. Lee, *SIBYL: A Tool for Managing Group Decision Rationale*, Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, CA, October 1990, pp. 79-92.

[Lee 92]
J. Lee, *Incremental Acquisition and Formalization of Design Rationlaes*, Working Notes of the AAAI 1992 Workshop on Design Rationale Capture and Use, AAAI, San Jose, CA, July 1992, pp. 179-188.

[Lemke 89]
A.C. Lemke, *Design Environments for High-Functionality Computer Systems*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, July 1989.

[Lemke, Fischer 90]
A.C. Lemke, G. Fischer, *A Cooperative Problem Solving System for User Interface Design*, Proceedings of AAAI-90, Eighth National Conference on Artificial Intelligence, AAAI Press/The MIT Press, Cambridge, MA, August 1990, pp. 479-484.

[Liskov, Zilles 74]
B.H. Liskov, S.N. Zilles, *Programming with Abstract Data Type*, SIGPLAN Notices, Vol. 9, No. 4, 1974, pp. 50-59.

[Lubars 89]
M.D. Lubars, *The IDeA Design Environment*, Proceedings of the 11th International Conference on Software Engineering, Pittsburgh, PE, May 1989, pp. 23-32.

[MacLean, Young, Moran 89]
A. MacLean, R. Young, T. Moran, *Design Rationale: The Argument Behind the Artifact*, Human Factors in Computing Systems, CHI'89 Conference Proceedings (Austin, TX), ACM, 1989, pp. 247-252.

[Mark, Schlossberg 90]
W. Mark, and J. Schlossberg, *A Design Memory Without Cases*, 1990.

[Marshall et al. 92]
C.C. Marshall, F.G. Halasz, R.A. Rogers, W.C. Janssen Jr., *Aquanet: A Hypertext Tool to Hold Your Knowledge in Place*, Hypertext '91 Proceedings, December 1992, pp. 261-275.

[McCall 91]
R. McCall, *PHI: A Conceptual Foundation for Design Hypermedia*, Design Studies, Vol. 12, No. 1, 1991, pp. 30-41.

[McCall et al. 90]
R. McCall, J. Ostwald, F. Shipman, N. Wallace, *The PHIDIAS HyperCAD System: Extending CAD with Hypermedia*, ACADIA Conference, 1990.

[McLaughlin 91]
S. McLaughlin, *Reading Architectural Plans: A Computable Model*, Technical Report, University of Sydney, 1991.

[McLaughlin 92]
S. McLaughlin, *The Term 'Function' and CAAD Research: Some Observations Drawn from Heidegger's Account of Equipment*, Technical Report, University of Sydney, 1992.

[McLaughlin, Gero 89]
S. McLaughlin, J.S. Gero, *Creative Processes - Can They Be Automated?*, Modeling Creativity and Knowledge-Based Creative Design (Reprints of the International Round-Table Conference: Modeling Creativity and Knowledge-Based Creative Design), Heron Island, Queensland, December 1989, pp. 69-94.

[Meyer 87]
B. Meyer, *Reusability: The Case for Object-Oriented Design*, IEEE Software, Vol. 4, No. 2, March 1987, pp. 50-64.

[Meyer 88]
B. Meyer, *Object-Oriented Software Construction*, Prentice Hall, Englewood Cliffs, NJ., 1988.

[Minsky 91]
M. Minsky, *Logical Versus Analogical or Symbolic Versus Connectionist or Neat versus Scruffy*, AI Magazine, Vol. 12, No. 2, Summer 1991, pp. 35-51.

[Miyake 86]
N. Miyake, *Constructive Interaction and the Iterative Process of Understanding*, Cognitive Science, Vol. 10, 1986, pp. 151-177.

[Miyake, Norman 79]
N. Miyake, D.A. Norman, *To Ask a Question, One Must Know Enough to Know What Is Not Known*, Journal of Verbal Learning and Verbal Behavior, Vol. 18, 1979, pp. 357-364.

[Nakakoji, Ostwald 92]
K. Nakakoji, J. Ostwald, *Reusing Design Rationale*, Working Notes of the AAAI 1992 Workshop on Design Rationale Capture and Use, AAAI, San Jose, CA, July 1992, pp. 199-206.

[Navinchandra 88]
D. Navinchandra, *Case-Based Reasoning in CYCLOPS*, Proceedings: Case-Based Reasoning

Workshop, J. Kolodner (ed.), Morgan Kaufmann Publishers, Clearwater Beach, FL, May 1988, pp. 286-301.

[Nielsen, Richards 89]
J. Nielsen, J.T. Richards, *The Experience of Learning and Using Smalltalk*, IEEE Software, May 1989, pp. 73-77.

[Nieper 85]
H. Nieper, *TRISTAN: A Generic Display and Editing System for Hierarchical Structures*, Technical Report, Department of Computer Science, University of Colorado, Boulder, CO, 1985.

[NKBA 92]
The National Kitchen and Bath Association, *Old Rules/New Rules or If You Want to be Successful at the Game... You Need to Know the Rules*, 1992.

[Norman 93]
D.A. Norman, *Things That Make Us Smart*, Addison-Wesley Publishing Company, Reading, MA, 1993, Expected publication, early 1993.

[Oppermann 92]
R. Oppermann, *Adaptively Supported Adaptability*, Sixth European Conference on Cognitive Ergonomics, Human-Computer Interaction: Tasks and Organization (Balatonfuered, Hungary), September 1992, pp. 255-270.

[Osterweil 87]
L. Osterweil, *Software Processes are Software too*, Proceedings of the 9th International Conference on Software Engineering (Monterey, CA), IEEE Computer Society, Washington, D.C., March 1987, pp. 2-13.

[Ostwald, Burns, Morch 92]
J. Ostwald, B. Burns, A. Morch, *The Evolving Artifact Approach to System Building*, Working Notes of the AAAI 1992 Workshop on Design Rationale Capture and Use, AAAI, San Jose, CA, July 1992, pp. 207-214.

[Owen 86]
D. Owen, *Answers First, Then Questions*, in D.A. Norman, S.W. Draper (eds.), *User Centered System Design, New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, pp. 361-375, ch. 17.

[Patel-Schneider 84]
P.F. Patel-Schneider, *Small Can Be Beautiful in Knowledge Representation*, AI Technical Report 37, Schlumberger Palo Alto Research, October 1984.

[Patel-Schneider, Brachman, Levesque 84]
P.F. Patel-Schneider, R.J. Brachman, H.J. Levesque, *ARGON: Knowledge Representation Meets Information Retrieval*, Fairchild Technical Report 654, Schlumberger Palo Alto Research, September 1984.

[Polanyi 66]
M. Polanyi, *The Tacit Dimension*, Doubleday, Garden City, NY, 1966.

[Pollack 85]
M.E. Pollack, *Information Sought and Information Provided: An Empirical Study of User/Expert Dialogues*, Human Factors in Computing Systems, CHI'85 Conference Proceedings (San Francisco, CA), ACM, New York, April 1985, pp. 155-159.

[Prieto-Diaz, Arango 91]
R. Prieto-Diaz, G. Arango, *Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press, Los Alamos, CA, 1991.

[Reder, Ritter 92]
L.M. Reder, F.E. Ritter, *What Determines Initial Feeling of Knowing? Familiarity With Question Terms, Not With the Answer*, Journal of Experimental Psychology: Learning, Memory, and Cognition, Vol. 18, No. 3, 1992.

[Redmiles 92]
D.F. Redmiles, *From Programming Tasks to Solutions -- Bridging the Gap Through the Explanation of Examples*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, 1992.

[Reeves 93]
B.N. Reeves, *The Role of Embedded Communication and Artifact History in Collaborative Design*, Dissertation Thesis, Department of Computer Science, University of Colorado, Boulder, CO, 1993, (forthcoming).

[Reisberg 87]
D. Reisberg, *External Representations and the Advantages of Externalising One's Thoughts*, Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA., July 1987.

[Repenning, Sumner 92]
A. Repenning, T. Sumner, *Using Agentsheets to Create a Voice Dialog Design Environment*, Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing, ACM Press, 1992, pp. 1199-1207, (also published as Technical Report CU-CS-576-92).

[Rich, Waters 86]
C.H. Rich, R. Waters (eds.), *Readings in Artificial Intelligence and Software Engineering*, Morgan Kaufmann Publishers, Los Altos, CA, 1986.

[Rittel 84]
H.W.J. Rittel, *Second-Generation Design Methods*, in N. Cross (ed.), *Developments in Design Methodology*, John Wiley & Sons, New York, 1984, pp. 317-327.

[Rittel, Webber 84]
H.W.J. Rittel, M.M. Webber, *Planning Problems are Wicked Problems*, in N. Cross (ed.), *Developments in Design Methodology*, John Wiley & Sons, New York, 1984, pp. 135-144.

[Salton, Buckley 90]
G. Salton, C. Buckley, *Improving Retrieval Performance by Relevance Feedback*, Journal of the American Society for Information Science, Vol. 41, No. 4, 1990, pp. 288-297.

[Schank 88]
R.C. Schank, *Remindings and Memory: Dynamic Memory: A Theory of Reminding and Learning in Computers and People (Chap2)*, Proceedings: Case-Based Reasoning Workshop, J. Kolodner (ed.), Morgan Kaufmann Publishers, Clearwater Beach, FL, May 1988, pp. 1-16.

[Schoen 83]
D.A. Schoen, *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York, 1983.

[Schoen 92]
D. A. Schoen, *Designing as Reflective Conversation with the Materials of a Design Situation*, Knowledge-Based Systems Journal, Vol. 5, No. 1, 1992, pp. 3-14.

[Seamail 92]
Software Engineer's Associates, *Report on Case Study in Using CASE Tools*, Seamail, Vol. 3, 1992, pp. 3-14, (in Japanese).

[Sharples 93]
M. Sharples, *Cognitive Support and the Rhythm of Design*, in T. Dartnall (ed.), *Artificial Intelligence and Creativity*, Kluwer Academic Publishers, Netherlands, 1993, (in press).

[Sheil 83]
B.A. Sheil, *Power Tools for Programmers*, Datamation, February 1983, pp. 131-143.

[Shipman 93]
F. Shipman, *Supporting Knowledge Base Evolution with Incremental Formalization*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, July 1993, Forthcoming.

[Silverman 92]
B.G. Silverman, *Survey of Expert Critiquing Systems: Practical and Theoretical Frontiers*, Communications of the ACM, Vol. 35, No. 4, April 1992, pp. 106-127.

[Silverman, Mezher 92]
B.G. Silverman, T.M. Mezher, *Expert Critics in Engineering Design: Lessons Learned and Research Needs*, AI Magazine, Vol. 13, No. 1, Spring 1992, pp. 45-62.

[Simon 81]
H.A. Simon, *The Sciences of the Artificial*, The MIT Press, Cambridge, MA, 1981.

[Simon 84]
H.A. Simon, *The Structure of Ill-structured Problems*, in N. Cross (ed.), *Developments in Design Methodology*, John Wiley & Sons, New York, 1984, pp. 145-166.

[Simon 91]
H.A. Simon, *Alternative Representations for Cognition: Search and Reasoning*, Technical Report, Department of Psychology, Carnegie Mellon University, 1991.

[Smithers et al. 89]
T. Smithers, A. Conkie, J. Doheny, B. Logan, K. Millington, *Design as Intelligent Behavior: An AI in Design Research Programme*, in J.S. Gero (ed.), *Artificial Intelligence in Design*, Springer-Verlag, Berlin, Germany, 1989.

[Snodgrass, Coyne 90]
A.S. Snodgrass, R. Coyne, *Is Designing Hermeneutical?*, Technical Report, Department of Architectural and Design Science, University of Sydney, Australia, 1990.

[Stahl 93]
G. Stahl, *Interpretation in Design: The Problem of Tacit and Explict Understanding in Computer Support of Cooperative Design*, Unpublished Ph.D. Dissertation, Department of Computer Science, University of Colorado, July 1993, Forthcoming.

[Steier 90]
D. Steier, *Creating a Scientific Community at the Interface Between Engineering Design and AI*, AI Magazine, Vol. 11, No. 4, 1990, pp. 18-22.

[Suchman 87]
L.A. Suchman, *Plans and Situated Actions*, Cambridge University Press, Cambridge, UK, 1987.

[Swartout, Balzer 82]
W.R. Swartout, R. Balzer, *On the Inevitable Intertwining of Specification and Implementation*, Communications of the ACM, Vol. 25, No. 7, July 1982, pp. 438-439.

[Thimbleby et al. 90]
H. Thimbleby, S. Anderson, I.H. Witten, *Reflexive CSCW: Supporting Long-term Personal Work*, in *Interacting with Computers*, Butterworth-Heinemann Ltd, 1990, pp. 330-336.

[Thompson, Croft 89]
R.H. Thompson, W.B. Croft, *Support for Browsing in an Intelligent Text Retrieval System*, International Journal of Man-Machine Studies, Vol. 30, 1989, pp. 639-668.

[Waters 85]
R.C. Waters, *The Programmer's Apprentice: A Session with KBEmacs*, IEEE Transactions on Software Engineering, Vol. SE-11, No. 11, November 1985, pp. 1296-1320.

[Williams 84]
M.D. Williams, *What Makes RABBIT Run?*, International Journal of Man-Machine Studies, Vol. 21, 1984, pp. 333-352.

[Winograd, Flores 86]
T. Winograd, F. Flores, *Understanding Computers and Cognition: A New Foundation for Design*, Ablex Publishing Corporation, Norwood, NJ, 1986.

[Wolverton, Hayes-Roth 91]
M. Wolverton and B. Hayes-Roth, *Proposal: Performing Invention with Analogies to "Far-Flung" Concepts*, Technical Report, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, May 1991.

[Wroblewski, McCandless, Hill 91]
D.A Wroblewski, T.P. McCandless, W.C. Hill, *DETENTE: Pratical Support for Practical Action*, Human Factors in Computing Systems, CHI'91 Conference Proceedings (New Orleans, LA), ACM, New York, 1991, pp. 195-202.

[Yakemovic, Conklin 90]
K.C.B. Yakemovic, and E.J. Conklin, *Report on a Development Project Use of an Issue-Based Information SYstem*, Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, CA, October 1990, pp. 105-118.

[Young, Greef, O'Grady 91]
R.E. Young, A. Greef, P. O'Grady, *SPARK: An Artificial Intelligence Constraint Network System for Concurrent Engineering*, in J. Gero (ed.), *Artificial Intelligence in Design'91*, Butterworth-Heinemann Ltd, Oxford, England, 1991, pp. 79-94.

Appendix A

User's Manual of KID

A.1. KIDSPECIFICATION

Figure 5-4 shows a typical screen image of KIDSPECIFICATION. The middle *Questions* window lists questions and optional answers. Designers can choose any answers by clicking on them. Designers can also select a question to consider by clicking on it. In both cases, the question is accumulated into the *Considered Question* window that keeps track of the history of the consideration. The summary of currently selected answers are presented in the *Current Specification* window. Each selected answer is accompanied with a weighting slider, with which designers can assign weights to prioritize the requirement items. The *Catalog* window on the left side lists names of the catalog examples. Designers can overview the specification of each of the examples. When designers click on one of the names, the specification of the named design appears in the *Current Specification* window.

Below is a description of major commands available in KIDSPECIFICATION.

- ***Create Question, Quick Question:*** allows designers to define a new question. The former provides a property sheet and the latter requires designers to interact through the *Command* window.
- ***Create Answer, Quick Answer:*** allows designers to define a new answer. The former provides a property sheet and the latter requires designers to interact through the *Command* window.
- ***Create Argument, Quick Argument:*** allows designers to define a new argument. The former provides a property sheet and the latter requires designers to interact through the *Command* window.
- ***Modify Question, Modify Answer, Modify Argument:*** allows designers to modify a question, an answer, or an argument through a property sheet.
- ***Select Issue:*** allows designers to attend to an issue (i.e., question) that has the specified name. The system scrolls the *Question* window if necessary, and highlights the question.
- ***Select Answer:*** allows designers to select the answer that has the specified name. The selected answer appears in the *Current Specification* window.
- ***Show Suggestions:*** lists suggestions (question-answer pairs) based on the currently selected answers. Suggestions are ordered according to numbers assigned representing their relative importance. An exclamation mark at the beginning of the presented answer indicates the occurrence of conflicts (i.e., suggesting another answer to the same question).
- ***Find Influenced Answers:*** lists suggestions that would be made if a certain answer is selected. This allows designers to play a what-if game in terms of an answer.
- ***Show Argument:*** presents associated answers and arguments to a specified question.

- ***Select Related Question:*** presents a related question to a specified suggestion (question-answer pair appeared in the *Suggestion* window) or to an argument.
- ***Show Further Argument:*** presents associated arguments to a specified suggestion.
- ***Set Options:*** allows designers to modify some default values of system's behavior, including the depth of suggestions, a mode whether to list all the computed suggestions each time one answer is selected, and the presentation style such as including authors of arguments.
- ***Copy; Store; Start New Specification:*** allows designers to make a copy the current specification, store it into a file, or to start from scratch with a name.
- ***Load Specification:*** loads the specification from the stored file
- ***Store Base Issues:*** saves the underlying argumentation base into a file.
- ***Save in Catalog:*** saves the current specification and construction into the catalog base. Designers can edit the values of attributes in CATALOGEXPLORER (see below).
- ***Resume Construction:*** brings designers to the CONSTRUCTION system.
- ***Catalog Explorer:*** brings designers to the CATALOGEXPLORER system.

A.2. CONSTRUCTION

Figure 5-5 shows a typical screen image of CONSTRUCTION. Designers can construct a design by selecting a design unit in the *palette* window, and locating it in the *work area* window. In the work area, designers can move, rotate, scale, and delete a design unit by using a mouse. Designers can copy a catalog example from the *catalog* window (bottom left) into the work area by clicking on it, and then modify it. The *Messages* window shows a list of critics fired.

Below is a list of major commands available in CONSTRUCTION.

- ***Critique (Praise) All:*** checks all the enabled critic rules, both generic and specific, applies them to the current construction, identifies violations (satisfying conditions), and presents critic messages.
- ***Critique (Praise) from Current Specification:*** checks the rules related to the current specification, identified by RULE-DELIVERER.
- ***Show Argument:*** brings designers to KIDSPECIFICATION and presents the associated argument from a specified fired critic.
- ***Show Rationale:*** brings designers to KIDSPECIFICATION and presents the associated specification (currently selected answer) from a fired specific critic.
- ***Identify Design Units:*** blinks design units for several seconds in the *Work Area* window associated with a specified fired critic.
- ***Set Options:*** allows designers to turn the delivery mode on and off, which determines whether the system automatically reorders the catalog elements in the catalog window when they pick a design unit in the palette.

- ***Edit Objects, List Objects, Edit Global Descriptions:*** allows designers to define a design object, such as a design unit (e.g., a microwave oven), or a relationship of configuration (e.g., facing to) using property sheets provided by MODIFIER, a system component that supports end-user modifiability. The detail description is found in Girgensohn.
- ***Resume Specification:*** brings designers to the KIDSPECIFICATION system.
- ***Catalog Explorer:*** brings designers to the CATALOGEXPLORER system.

A.3. CATALOGEXPLORER

Figure 5-7 shows a screen image of CATALOGEXPLORER, which supports designers in searching the catalog space. The *Matching Designs* window lists the names of the currently retrieved examples from the catalog. By clicking on one of them, designers can view the example, which consists of (1) the construction (a floor plan constructed in CONSTRUCTION), (2) the specification (question-answer and weight pairs specified in KIDSPECIFICATION), and (3) the attributes, as presented in the three windows in the middle. The *Bookmark* window holds the names of the catalog examples that interested designers. Designers can place a presently displayed example by using the *Add to Bookmark* command.

Below is a list of major commands available.

- ***Retrieval by Matching — Construction:*** allows designers to select some of characteristics observed in the current construction (as shown in the *Current Construction* window on the right side) presented in a menu form and to retrieve catalog examples that have the same set of selected characteristics.
- ***Retrieval by Matching — Specification:*** allows designers to select some of the currently specified items (as shown in the *Current Specification* window on the right side) presented in a menu form and to retrieve catalog examples that have the same set of specifications.
- ***Retrieve from Query:*** retrieves catalog examples by a specified query consisting of restriction in terms of attributes of the example.
- ***Order by Specification:*** invokes CASE-DELIVERER to order the catalog examples either of the whole catalog base or of the currently retrieved ones using delivery rationale inferred from the current specification.
- ***Show Delivery Rationale:*** lists conditions that CASE-DELIVERER used to compute the values for ordering.
- ***Evaluate Example:*** allows designers to evaluate the example in terms of the current specification with four options: *Critique the Example*, *Praise the Example*, *Critique the Example from Current Specification*, and *Praise the Example from Current Specification.*
- ***Add to Bookmark:*** adds the currently displayed catalog example in the *Bookmark* window.
- ***Switch Display:*** allows designers to select one of four displays: current construction, current specification, a hierarchical structure of the catalog base, or currently specified queries.

- ***Resume Construction:*** brings designers to the CONSTRUCTION system.
- ***Resume Specification:*** brings designers to the KIDSPECIFICATION system.

Appendix B

Representation of Catalog Examples

Below are two representaitons of a catalog example (*"Elly-Kitchen"*) shown in Figure 5-7. A catalog example has two representaitons: construction and specification. Currently, the information about the construction of a catalog example and that of the specification are stored separately.

Construction:.

```
(("Elly-Kitchen" :descriptions
  ((prerequisite (rinses ? dishes) (cleans ? dishes))
   (prerequisite (cleans ? food) (cooks ? food))
   (flammable curtains) (prerequisite (cleans ?x food) (cooks ?y food))
   (prerequisite (rinses ?x dishes) (cleans ?y dishes)))
  :svpairs
  ((author "Elly")
   (creation-date "11/30/92")
   (modification-date "11/30/92")
   (shape "CORRIDOR")
   (style nil)
   (annotation "This is a small but neat kitchen.")
   (example-type nil))
  :categories (kitchen))
 (horizontal-wall
   :left-x 7.5 :top-y 0 :width 156 :depth 6
   :rotation 0
   :descriptions nil :unique-id 1)
 (four-element-asymmetric-stove
   :left-x 267/2 :top-y 6 :width 30 :depth 24
   :rotation 0 :descriptions  nil  :unique-id 2)
 (base-cabinet :left-x 109.0 :top-y 6 :width 24 :depth 24
               :rotation 0 :descriptions  nil :unique-id 3)
 (single-door-refrigerator :left-x 96.5 :top-y 63 :width 36 :depth 26.5
                           :rotation 180 :descriptions
                           nil :unique-id 4)
 (vertical-wall :left-x 327/2 :top-y 0 :width 90 :depth 6
                :rotation 90 :descriptions nil :unique-id 5)
 (horizontal-wall :left-x 0 :top-y 90 :width 169.5 :depth 6
                  :rotation 0 :descriptions nil :unique-id 6)
 (double-bowl-sink :left-x 75.0 :top-y 6.0 :width 33 :depth 24
                   :rotation 0 :descriptions   nil  :unique-id 7)
 (base-cabinet :left-x 267/2 :top-y 65.5 :width 30 :depth 24
               :rotation 0 :descriptions  nil :unique-id 8)
 (base-cabinet :left-x 68.5 :top-y 65.5 :width 27 :depth 24
               :rotation 0 :descriptions  nil :unique-id 9)
 (vertical-wall :left-x 1.5 :top-y 0 :width 33 :depth 6
                :rotation 90 :descriptions nil :unique-id 10)
 (dishwasher :left-x 49.5 :top-y 6 :width 24 :depth 24
             :rotation 0 :descriptions  nil :unique-id 11)
 (base-cabinet :left-x 7.5 :top-y 6.5 :width 21 :depth 24
               :rotation 0 :descriptions nil  :unique-id 12)
 (base-cabinet :left-x 30 :top-y 6.5 :width 18 :depth 24
               :rotation 0 :descriptions
  nil :unique-id 13))
```

Specification:.

```
(:name elly-kitchen
 :type kitchen
 :base-issue-version-number 57
 :selected-issue-ans
 (("size-of-family" (("one" 10)))
  ("need-dishwasher" (("yes" 9)))
  ("entertainment-requirement" (("yes" 2)))
  ("how-many-meals" (("once" 5)))
  ("shape-of-kitchen" (("corridor" 4))))
 :selected-args nil :full-text nil)
```